08-14-00

JC883 U.S. PTO 08/11/00

JC877 U.S. PTO 09/637821 08/11/00

Please type a plus sign (+) inside this box → [+]

PTO/SB/05 (4/98)
Approved for use through 09/30/2000. OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 C.F.R. § 1.53(b))

Attorney Docket No.	PACIF-55288
First Inventor or Application Identifier	Johnson et al.
Title	COMPENSATION SYSTEM AND METHOD FOR . .
Express Mail Label No.	EL 590 182 529 US

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO: **Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231**

1. [ ] * Fee Transmittal Form *(e.g., PTO/SB/17)* *(Submit an original and a duplicate for fee processing)*
2. [X] Specification [*Total Pages* 44] *(preferred arrangement set forth below)*
   - Descriptive title of the Invention
   - Cross References to Related Applications
   - Statement Regarding Fed sponsored R & D
   - Reference to Microfiche Appendix
   - Background of the Invention
   - Brief Summary of the Invention
   - Brief Description of the Drawings *(if filed)*
   - Detailed Description
   - Claim(s)
   - Abstract of the Disclosure
3. [X] Drawing(s) *(35 U.S.C. 113)* [*Total Sheets* 15]
4. Oath or Declaration [*Total Pages*]
   - a. [ ] Newly executed (original or copy)
   - b. [ ] Copy from a prior application (37 C.F.R. § 1.63(d)) *(for continuation/divisional with Box 16 completed)*
     - i. [ ] DELETION OF INVENTOR(S) Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. §§ 1.63(d)(2) and 1.33(b).

** NOTE FOR ITEMS 1 & 13: IN ORDER TO BE ENTITLED TO PAY SMALL ENTITY FEES, A SMALL ENTITY STATEMENT IS REQUIRED (37 C.F.R. § 1.27), EXCEPT IF ONE FILED IN A PRIOR APPLICATION IS RELIED UPON (37 C.F.R. § 1.28).*

5. [ ] Microfiche Computer Program *(Appendix)*
6. Nucleotide and/or Amino Acid Sequence Submission *(if applicable, all necessary)*
   - a. [ ] Computer Readable Copy
   - b. [ ] Paper Copy (identical to computer copy)
   - c. [ ] Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

7. [ ] Assignment Papers (cover sheet & document(s))
8. [ ] 37 C.F.R.§3.73(b) Statement *(when there is an assignee)* [ ] Power of Attorney
9. [ ] English Translation Document *(if applicable)*
10. [ ] Information Disclosure Statement (IDS)/PTO-1449 [ ] Copies of IDS Citations
11. [ ] Preliminary Amendment
12. [X] Return Receipt Postcard (MPEP 503) *(Should be specifically itemized)*
13. [ ] * Small Entity Statement(s) *(PTO/SB/09-12)* [ ] Statement filed in prior application, Status still proper and desired
14. [ ] Certified Copy of Priority Document(s) *(if foreign priority is claimed)*
15. [ ] Other: .....

16. **If a CONTINUING APPLICATION,** *check appropriate box, and supply the requisite information below and in a preliminary amendment:*
    [ ] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No: 60/148,412
    Prior application information: Examiner _____ *Group / Art Unit:* _____

For CONTINUATION or DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 4b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

17. CORRESPONDENCE ADDRESS

[XX] *Customer Number or Bar Code Label* *(Insert Customer No. or Attach bar code label here)* or [ ] *Correspondence address below*

Name	David S. Sarisky, Esq. 24201 PATENT TRADEMARK OFFICE				
Address					
City		*State*		*Zip Code*	
Country		*Telephone*		*Fax*	

Name (Print/Type)	David S. Sarisky, Esq.	*Registration No (Attorney/Agent)*	41,288
Signature		*Date*	8/11/00

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

APPLICATION

of

Keith O. Johnson

Mats Myrberg

Timothy E. Onders

and

Alexander J. Limberis

for

UNITED STATES LETTERS PATENT

on

COMPENSATION SYSTEM AND METHOD FOR SOUND REPRODUCTION

Docket No. PACIF-55288
Sheets of Drawings: Fifteen (15)
[179710.1]

Attorneys
FULWIDER PATTON LEE & UTECHT, LLP
Howard Hughes Center
6060 Center Drive, Tenth Floor
Los Angeles, CA 90045

EXPRESS MAIL LABEL NO. EL 590 182 529 US

COMPENSATION SYSTEM AND METHOD FOR SOUND REPRODUCTION

RELATED APPLICATION

This application claims the benefit of co-pending provisional application serial no. 60/148,412 filed August 11, 1999.

BACKGROUND OF THE INVENTION

Field of the Invention:

This invention relates generally to a compensation method and system for use in sonic transmission and reproduction systems and more particularly to a compensation method and system that uses parametric values to control or adjust processes having transforms or models with properties or responses like the components or elements used in the transmission or reproduction system.

Description of Related Art:

Most audio reproduction systems use electromechanical loudspeakers to acoustically reproduce audio signals. The electrical, mechanical, and acoustical properties of the loudspeakers are often less than ideal, causing distortions, response anomalies, and other coloration of the sound. Many techniques are used to compensate for the loudspeaker's characteristics in order to improve perceived audio quality.

Functional or behavioral models of loudspeakers are used in practice and found in literature to develop such compensations. A good example of models and how the modeling process works is described in "Active Equalization of Loudspeakers", Speaker Builder, February 1997. Models consolidate technical languages and are usually intended to imitate or simulate the acoustic responses of the speaker system from electrical stimulus. Model creation or synthesis frequently begins by making functional groupings of elements which collectively represent or behave like all or part of the speaker. Coil and magnet parts become motors, which are represented by resistors, inductors, capacitors, back EMF generators and other transformed parts. A combination of factors such as air volume, moving mass, acoustic loading, magnetic-braking, and mechanical losses might be analyzed and simplified to LCR resonator networks or circuits. Most often, the transformed electromechanical, acoustic, and mechanical representations expressed in the model are further simplified or reduced

to fewer elements. The model still responds like the speaker, but the parts making the model no longer have exact behavioral equivalence to the parts making the speaker. Consequently, traditional models are neither intended, nor capable of making parametrically addressed zero-phase compensations when speaker parts are changed.

One could characterize and invert the frequency response, as well as other properties, of a well-conceived model and achieve linear-phase correction of the loudspeaker. The technique does work to a fashion, but its dedicated, inflexible circuitry or specialized process tied to the traditional model limits its use to a one-speaker design. Some high-quality crossover networks constructed to divide the signal spectrum amongst multiple drivers may have some conjugate response correction like this.

A low-frequency resonant boost is intentionally designed for most speakers. Frequently, traditional models are made to represent quantifiable and predictable acoustic behavior as well as other speaker design factors affecting bass response. Mechanical construction and properties of air determine frequency, resonant losses and the configuration's effect on acoustic output from the speaker. A good approximation to a zero-phase conjugate or same-order correction for a wider frequency range can be designed and implemented in this manner. Several components are needed to match the resonant behavior, but all interact with each other when adjustments are made for a different speaker of similar concept and design. Therefore, the operation is not strictly parametrically controlled, as the adjustments must be re-calculated from the model to create the minimum-phase or exact match needed for best fidelity with the new speaker. When more corrections are added the interaction problem becomes formidable. The system must be tuned experimentally or the model analyzed each time an adjustment is made. Consequently, lumped model processes for response flattening are inherently designed for a specific speaker. The process must be redesigned for other speakers.

Traditional curve-fitting methods can require hundreds of data points and corresponding adjustments to set up and many components or much processing power to match the acquired frequency response. Analog methods are impractical and digital processes require much computation and extensive architectures to do this. Neither can provide phase accurate responses or the hidden corrections described later without having knowledge of the speaker and its operation. Without a model, the effort to

combine amplitude, time and phase corrections together from measured responses becomes formidable.

Some of the most important behaviors of loudspeakers (with respect to acoustically perceptible effect) cannot be modeled or implemented from traditional methods. Such behaviors include standing wave interference, modal breakup, and coupled resonance as well as nonlinear consequences from such potentially interacting acoustic and mechanical behaviors. Counterproductive random motion or breakup may occur. Even when the average response remains flat or is the same as other frequencies being reproduced, energy can build up during signal stimulus and be released when the signal changes or ceases. In addition, other spatial factors related to stiffness of moving parts and high frequency de-coupling for motions away from a driving voice coil need to be considered. Any of these can create source movements, delayed energy release, and phase error to binaural hearing. Often, such destructive responses can be invisible or very difficult to interpret from traditional microphone-and-spectrum-analyzer calibration methods.

For example, unwanted responses arising from nodal and standing wave behavior affect the settling time, directional behavior, and radiated output of a speaker. Frequently, these responses cause perceptual changes to intelligence signals yet may not be visible or recognizable from response plots. Mechanical motions having large stored energy can be out of phase at different parts of the transducer. The acoustic output might appear to be flat, but human binaural hearing can localize the behavior to its source and the altered perception can degrade stereo imaging.

Often, mechanical disturbances are audible yet invisible or hard to interpret from response measurements made using a frequency sweep and microphone. Parts of a radiating surface can vibrate with different phase relationships to other parts, so that their additive acoustic output is low compared to motions within the transducer and the energy storage involved. When signals at the node frequency change and suddenly stop the release of stored energy can interact with other signals at different frequencies. The resulting beat sounds between the two frequencies can be audible and very objectionable. Sounds with spectra in the interference frequency range may appear louder and granular. Human binaural hearing can localize the disturbance to the driver or surfaces from which directional lobes might bounce, thereby imparting further damage to the stage illusion from multi-speaker stereo reproduction. For this situation, frequencies creating the mechanical disturbance must be sufficiently

attenuated to prevent unmasked reproduction of consequential responses. Experience has shown that a sharp, deep notch needed to do this removes enough energy around the correction frequencies to cause a nasal sound. If this inappropriate correction is modified to achieve flat response, then the mechanical sounds remain along with a potential undesirable balance aberration.

Many small loudspeakers are constructed with a transducer, enclosure, and some resonant means to extend bass response such as a port or passive radiator. Usually, these parts are designed to achieve a practical and economical compromise between efficiency, frequency response accuracy, bass extension, and acceptable distortion. Designers of inexpensive, low-powered systems generally opt for higher efficiency to reduce amplifier requirements along with related costs of power supplies and packaging. The compromise situation exposes many undesirable behavioral aspects.

Most traditional speaker correction methods apply some variation of amplitude equalization to flatten and extend response from speakers. Adjustments are sometimes done by ear. To be quantitative, one must acquire relevant data. The most common techniques to do this use spectral analysis from noise stimulus. Then, response plots or displays indicate how an equalizer is to be adjusted. More sophisticated techniques based on delayed acceptance or sampled windows can measure first-arrival responses from the speaker and remove higher-frequency room disturbance to create anechoic-like data. The intent is to capture information relevant either to a listener in a room or to standard measurement practice where a test microphone is usually specified and placed one meter from the speaker. Such technique creates a response that may sound balanced to the single-point test microphone. One or more known systems go slightly beyond this by adjusting path lengths, or time delays to align multiple speakers to a listening position.

Other techniques provide transient response waveforms, waterfall or successions of spectral plots after an event. Group delay and time-related information is acquired. Such data needs interpretation and has limited use for frequency response leveling practice. Some behavioral responses can be recognized but much more information must be known about the speaker. Measurement devices such as accelerometers, differential acoustic probes, as well as microphones, are needed for this. Instrumentation may be placed near a suspected behavior site and moved to explore how a response changes with position. Weighted notches can be tuned or

slowly swept through suspected frequencies while subjectively observing noise production. More information is needed about dimensions of parts, listening positions, as well as floor, shelf, possibly a computer monitor, or other interceding objects that may be part of the listening environment. Other technical specifications or expressions are needed to complete the conjugate model capable of time-phase-accurate correction.

A human operator can assume an alignment role by adjusting a graphic equalizer, manually tuning a parametric filter, or changing settings to a crossover device. Commercial analog components perform these functions, but they have limitations. Graphic equalizers have up to 31 bands or resonators, parametric devices include several adjustable filters and a few have variable crossover and shelf functions. Many more filters are needed. Combinations of graphic and parametric equalizers are incapable of providing a large enough number of points, nor the exact phase and time response to effectively compensate complex behavior from a loudspeaker. Either the corrections do not match specific frequencies, thereby creating phase error, or the number of filters is inadequate to deal with settling time and standing wave issues. Group delay distortion, time-phase error, incomplete correction and other shortcomings are likely to outweigh other improvements.

DSP filters can create many more filter sections than is practical from analog circuits. Graphic equalizers made up with parametrically controlled sections have been used with specialized control-generating software to create room response leveling. Such processes are difficult to set up because the room interferes with the identification of important behavioral indicators. Without their input, conjugate response corrections are not possible. Standing wave and nodal distortion corrections could be made from such a system. However, the awkward compiling and processing needed to parametrically move the compensated notches would be difficult. Most likely, a single point response pickup and FFT has been used for data input to the system. Such methods cannot respond to or provide the time-phase information needed to create a true conjugate response to the speaker. Analysis systems, such as MLSSA, can remove room interference from measurements, and can produce frequency, transient, and settling response data from a loudspeaker system. However, the large amount of data from these measurements must be interpreted. The multiple-band graphic equalizer is not a good choice to install the correction.

DSP systems can economically create many parametric filters and time-related processes that are impractical with analog circuitry. Traditional large-scale DSP systems have little means to identify and cull out speaker behavior from other measurement anomalies. Their frequency-domain responses are likely to add phase errors and to overlook delayed settling energy. The sound might improve for one listening position but it will degrade for all others. More likely, the reproduced sound will change without definitive improvement.

Hence, those concerned with the reproduction of sound have recognized the need for a system and method of modeling the complete behavior of sound reproduction devices such that conjugate responses to the sound reproduction device responses may be created. The need for a system and method employing modifiable conjugate response has also been recognized. Furthermore, the need has also been recognized for a system and method that compensates the reproduction of sound independent of the environment in which the sound is to be heard. The present invention fulfills these needs and others.

SUMMARY OF THE INVENTION

Briefly, and in general terms, the present invention provides a system and method for modeling individual response characteristics of a sonic reproduction device to create a conjugate model for improving frequency, time, phase, and amplitude performance of the device and to provide improved sonic balance, sound clarity, reduced distortion and improved stereo imaging.

In a first aspect, the invention relates to an apparatus for modifying an electrical audio signal for input to a sonic reproduction device characterized by a plurality of individual responses. The individual responses of the device combine to define an overall response. Each individual response includes one or more of a frequency, time, phase or transient response. The apparatus includes a plurality of modification filters having modification responses that simulate the plurality of individual responses of the sonic reproduction device. The modification filters receive the electrical audio signal, modify the electrical audio signal and provide the electrical audio signal to the sonic reproduction device. The apparatus further includes a plurality of adjustable parameters. Each of the adjustable parameters is associated with at least one of the modification filters. The adjustable parameters allow for adjustments to the responses of the modification filters. The adjustments create a

plurality of individual conjugate responses. Each individual conjugate response is associated with at least one of the plurality of individual responses.

By creating a plurality of filters or networks having responses that model the individual responses of the reproduction device and providing parameters for adjusting the filter or network responses, the system allows for the creation of conjugate responses that provide specific opposing or correction responses to the response of the reproduction device.

In a detailed aspect of the invention, the plurality of individual responses of the sonic reproduction device are related to at least one of mechanical, acoustic and electromagnetic behavior of the sonic reproduction device. In another detailed facet of the invention, the plurality of modification responses combine to form an overall response that is a conjugate to the overall response of the sonic reproduction device.

In yet another detailed aspect of the invention, at least one of the modification filters comprises a cut-off filter and the parameters for adjusting the frequency response of the cut-off filter include peak frequency, amplitude and Q parameters. In still another aspect of the invention, at least one of the modification filters comprises a constant slope equalizer and the parameters for adjusting the frequency response of the constant slope equalizer include crossover frequency and boost shelf parameters. In other detailed facets of the invention, at least one of the modification filters comprises a parametric notch filter and the parameters for adjusting the frequency response of the parametric notch filter include notch frequency, amplitude and Q parameters and at least one of the modification filters comprises a parametric notch-boost filter and the parameters for adjusting the frequency response of the parametric notch-boost filter include notch frequency, amplitude and Q parameters.

In a second aspect, the invention relates to a sound compensation system for altering an electrical audio signal for input to a sonic reproduction device having associated behavioral characteristics. The system includes a model of the sonic reproduction device. The model includes a plurality of filters or processes that simulate at least one of the behavioral characteristics of the sonic reproduction device. Each filter has an associated response that combine to define an overall response for the model. Each individual response includes one or more of a frequency, time, phase or transient response. The system also includes a controller that modifies the response of each of the plurality of filters to transform the filter into a conjugate filter. Each

conjugate filter has a response that is a conjugate to the original response of the filter or process.

In a detailed aspect of the invention, the behavioral characteristics are defined by individual or groups of individual components of the sonic reproduction device. By modeling the reproduction device's individual components and the characteristics of those components or groups of components, individual compensations for these characteristics can be created and manipulated parametrically. Therefore, these same compensations can be applied to additional systems having similar components or characteristics.

In other detailed aspects of the invention, the filters are defined by digital signal processes or by analog circuits and the controller includes a computer or adjustable circuit components. In other detailed aspects of the invention, the sonic reproduction device comprises a speaker and at least one of the plurality of filters includes at least one associated adjustable parameter and the value of the parameter is calculated based on physical characteristics of the speaker, derived from a standard speaker model or determined experimentally using standard test measurements. In yet another detailed aspect of the invention, the controller is configured such that an adjustment in the setting of one parameter modulates the setting of at least one other parameter. In still another detailed facet of the invention, the controller monitors the program conditions at the sonic reproduction device and sets at least one of the parameter values based on the program conditions.

In a third facet, the invention relates to a sound system. The sound system includes a sonic reproduction device having associated mechanical, acoustic and electromagnetic behavioral characteristics. The sound system also includes a source for outputting an electrical audio signal to a model of the sonic reproduction device. The model includes a plurality of filters that simulate at least one of the mechanical, acoustic and electromagnetic behavioral characteristics of the sonic reproduction device. Each filter has an associated response that includes at least one of a frequency, time, phase or transient response. The model outputs the electrical audio signal to the sonic reproduction device. The sound system further includes a controller that modifies the responses of the filters to transform the model into a conjugate model having a plurality of filters with responses that comprise conjugates to the original response of the filter.

In a fourth aspect, the invention relates to a method for modifying an electrical audio signal for input to a sonic reproduction device that is characterized by a plurality of individual responses which in combination define an overall frequency response for the sonic reproduction device. Each individual response includes one or more of a frequency, time, phase or transient response. The method includes the steps of simulating the plurality of individual responses with a plurality of filters and adjusting the responses of the plurality of filters such that, for each filter, the adjusted response comprises a response that is a conjugate to one of the individual responses. The method further includes the step of inputting the electrical audio signal to the filters.

In a detailed aspect of the invention, at least one of the filters comprises a cut-off filter and the step of adjusting the frequency response of the cut-off filter includes the step of setting at least one of peak frequency, amplitude and Q. In another detailed facet of the invention, at least one of the filters comprises a constant slope equalizer and the step of adjusting the frequency response of the constant slope equalizer includes the step of setting at least one of crossover frequency and boost shelf. In still another detailed aspect, at least one of the filters comprises a parametric notch filter and the step of adjusting the frequency response of the parametric notch filter comprises the step of setting at least one of notch frequency, amplitude and Q. In yet another detailed aspect of the invention, at least one of the filters comprises a parametric notch-boost filter and the step of adjusting the frequency response of the parametric notch-boost filter comprises the step of setting at least one of notch frequency, amplitude and Q.

In a fifth facet, the invention relates to a method of altering an electrical audio signal for input to a sonic reproduction device having associated behavioral characteristics. The method includes the step of simulating at least one of the behavioral characteristics of the sonic reproduction device with a plurality of filters. Each of the filters has an associated response comprising at least one of a frequency, time, phase or transient response. The method further includes the step of, for each of the filters, modifying the response of the filter to transform the filter into a conjugate filter having a response that comprises a conjugate to the original response of the filter.

In detailed facets of the invention, the sonic reproduction device comprises a speaker, at least one of the plurality of filters has at least one associated adjustable parameter and the step of modifying the response of the filter includes one or more of

the following pairs of steps: calculating the value of the adjustable parameter value based on the physical characteristics of the speaker and setting the parameter to the calculated value, deriving the adjustable parameter from a standard speaker model and setting the parameter to the derived value and determining the adjustable parameter experimentally using standard test measurements; and setting the parameter to the determined value. In still another detailed facet of the invention, the method further includes the step of modulating the setting of at least one parameter in response to the setting of another parameter. In another detailed facet, the method further includes the steps of monitoring at least one program condition at the sonic reproduction device and setting at least one of the parameter values based on the program condition.

These features and the ability to maintain sonic neutrality make a compensation system and method capable of complex, dynamically changing response corrections, which can be controlled and adjusted with simplified, intuitive control specifications. Compared to traditional response-leveling methods, the compensation method and system requires less processing complexity and can easily be applied to different sound-reproducing systems. The compensation method and system, as a whole or in piece parts, can be turned on or off, moved from one frequency to another, or otherwise be changed by simple, intuitive commands.

These and other aspects and advantages of the present invention will become apparent from the following more detailed description, when taken in conjunction with the accompanying drawings which illustrate, by way of example, the preferred embodiments of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a sonic reproduction system incorporating a compensation system in accordance with the invention;

FIG. 2 is a block diagram of a compensation system having a plurality of modification filters, each having a plurality of adjustable parameters for modifying one or more of the frequency, time and phase responses of the filters;

FIGS. 3a-3b depict a plurality of response components for a loudspeaker;

FIG. 3f depicts the overall response formed when combining the individual responses of FIG. 3a-3e;

FIGS. 4a-4e depict a plurality of adjustable modification responses for a compensation system, each response is adjustable to form a conjugate response to the response components of FIGS. 3a-3e;

FIG. 4f depicts the overall conjugate response formed when combining the individual frequency responses of FIGs 4a-4e;

FIG. 5 is a graph depicting the motion for a circular cone to produce one acoustic watt output;

FIG. 6 depicts the frequency responses for two small loudspeaker drivers;

FIG. 7 depicts a waterfall plot wherein the arrows at the right show the increasing time axis;

FIGS. 8a-8f depicts a series of graphical user interfaces for adjusting parametric controls for modifying the responses of the modification filters;

FIG. 9 is a schematic of an low-pass/high-pass peaking filter, where all Cs must change to move frequency;

FIG. 10 is a schematic of an active RC or constant slope equalizer that boosts and has approximate parametric independence, where Cc and Rc are both very large to bias the op amp;

FIG. 11 is a schematic of a frequency movable notch;

FIG. 12 is a schematic of an alternate configuration of a frequency movable notch;

FIG. 13 includes a schematic diagram of a weighted notch filter and the responses for boost, and notch components and a combined response, obtainable using the filter;

FIG. 14 includes a schematic diagram of a multi-resonant weighted notch filter and the responses for boost and notch components and a combined response obtainable using the notch filter;

FIG. 15 depicts the frequency, and phase responses for a notch filter;

FIG. 16 is a schematic of a delayed interference simulator/compensator where CW equals the same response as interference, CCW equals a conjugate correction and RC equals a decrease compensation for higher frequencies; and

FIG. 17 is a schematic of an all-pass or phase shift network.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The following description focuses on the application of the present invention to a loudspeaker system. The invention, however, is not limited to such applications and may be applied to other sonic transmission and reproduction devices such as those set forth at the end of the specification.

Referring now to the drawings, wherein like reference numerals denote like or corresponding parts throughout the drawing figures, and particularly to FIG. 1, there is shown a system 10 incorporating the present invention. The system 10 includes a signal source 12, for providing an electrical audio signal. The signal source may be, for example, a CD player. The output 14 from the signal source 12 is input to a compensation system 16 which employs detailed features and aspects of the present invention. The output 14 is processed by the compensation system 16 to produce a compensated electrical audio signal 18 which is fed to a power amplifier 19 and a loudspeaker 20.

With reference to FIG 2, the compensation system 18 employs a plurality of filters 24 which, either individually or in combination, exhibit individual compensation responses which simulate the frequency, time and phase responses exhibited by the various mechanical, acoustic and electromagnetic components of the loudspeaker. Associated with each of the filters 24, and thus each of the compensation responses, is one or more adjustable parameters 22. The filters 24 are created using DSP or analog circuits. Digital signal processing is the preferred implementation, since analytical models of loudspeaker behavior translate easily to the mathematical synthesis techniques used for designing digital systems. Analog circuits have distortion build-up in cascade architectures. Therefore, a practical analog implementation of the compensation system 18 is possible only when the analog designer is aware that parallel-connected circuit elements can be used when behavioral responses are isolated and non-interactive. Some parameter 22 adjustments interact with one or more parts of the compensation system. A digital implementation provides the ability to utilize a compilation of possible settings or to calculate these relationships as needed. When the compensation system is controlled from a computer the setup adjustments appear simple and intuitive.

Each of these computed and adjustable compensation responses can be scaled to a parametrically variable, feature, or design aspect, relating to size, a moving property, or acoustic radiation behavior, amongst other things. Time delays,

maximum excursion limits, wavelength relationships, nodal and standing wave frequencies, boundary reflections and like properties of the speaker can be adjusted and used by the compensation system. Instead of compiling or reducing speaker elements to create a lumped response system, the compensation system uses minimum-phase equivalence and strings of non-interacting filters.

A typical overall response for a small speaker is shown in FIG. 3f. The same response is repeated at the top of FIG. 4. Figures 3a-3e shows a plurality of individual responses related to speaker components which combine to produce the overall response curve. Figures 4a-4e show a plurality of individual compensation responses which are adjusted via a plurality of adjustable parameters, *e. g.*, L_S, H_S, L_X, H_X, etc., to produce a plurality of conjugate responses. Properties of the parameters are described in detail below. As shown in FIG. 4f, the individual conjugate responses combine to produce or an overall conjugate response . The combination of the overall speaker response (FIG. 3f) and conjugate response (FIG. 4f) produce a smooth response.

From FIGS. 3a and 4a, the first and second cutoff inflections for both high and low frequencies are evident. These points on the curve determine the frequencies for L_X, L_S, H_X, and H_S. When voice coil length and motional compliance capability are known, then a calculation or chart, similar to the one shown in FIG. 5, can help determine a reasonable frequency and amplitude for L_P. H_P is treated in a similar manner. Related inflections, cutoff, and frequencies are shown in FIGS. 3b and 4b. Amplitude settings of L_P and H_P have subjective power handling and weighted compensation issues.

Self-resonance of the driver operating in its enclosure can be measured or calculated to yield W_0. The related Q_0 and $|A|_0$ settings can be experimentally measured or calculated from traditional models, provided the acoustic coupling factor has been removed. Figures 3d and 4d show this bass compensation.

Representations of a mechanical resonance, like W_1, are seldom used in conventional models. The higher frequency parts of FIGS. 3d and 4d show that acoustic response error and coloration from W_1 can be removed by setting $|A|_1$. A surround resonance is noted and a preset or default Q can be chosen to compensate a resonant behavior model typical of loudspeaker materials involved.

Nodes and interference behaviors are evident in FIG. 3e. Other examples in FIG. 6 reveal node jumps, interference, and related problems frequently called "cone

cry" because of harsh, smeared sound. Waterfall or MLSSA plots like those shown in FIG. 7, can show the frequency and delayed breakup consequences typical of such high-energy moving interference. Hidden compensation notches W_{CC1} and W_{CC2} of FIG. 4e remove these frequencies and energy storage problems.

A slope or tilt EQ is added to achieve a best listener preference. Figures 3c and 4c show behavior and correction. Additional downward tilt might be added for subjective balance.

As can be seen, approximately 12 to 16 adjustments, each spanning as few as 8 bits of resolution, can make a very good conjugate correction. Representative frequency responses from two loudspeaker driver units are shown in FIG. 6. Note that, as indicated by the roll off of the response curve at the low end, the W_0 and second inflection response related to L_P are missing because the driver operates in a very large enclosure. Other than this, the other parametrically related features are evident and adjustments can be worked out from these curves.

The following parameters allow for the creation of, simulations of, or conjugates to, the complex frequency, phase, and time responses of a loudspeaker. These adjustable parameters 22 and their operations can approximate zero-phase response, extend bass output, and remove mechanical sounds from a speaker (other parameters can be used in a similar manner). The individual compensations can be performed digitally or using active or passive analog circuits, such as RC circuits, analog resonators, or fully parametric circuits like state-variable filters or biquads.

Term	Acronym	Description, Relationships
Low Crossover	L_X	Radiating area, compliance, air volume, acoustic coupling, and B_L factors. (first inflection@+/- 6dB per octave)
High Crossover	H_X	Mass, radiation area relationships, stiffness, B_L factors. (First inflection at +/- 6dB per octave)
Low Boost Shelf	L_S	Design limit, sonic balance, (XdB stops slope)
High Boost Shelf	H_S	Design Limit, power handling, useful response linearity (XdB stops slope)

Term	Acronym	Description, Relationships
Low Peaking	W_L	Boost for second coupling inflection. Hidden compensation for high-pass or low frequency limit filter. (default cutoff slope, XdB peak)
High Peaking	W_H	Boost for second high-frequency loss inflection. Hidden compensation for low-pass or high-frequency limit filter. (default cutoff slope, XdB peak)
Motor	B_L	Force, back EMF, inductance factors. Modulates responses and effects of parametric values
Bass Resonance	W_0	Volume, compliance, area, damping, B_L factors (frequency, Q, amplitude)
Port Tuning	W_B	Bass tune model (traditional) (input for double-tuned filter model)
Mechanical Resonance	W_1, W_2, through W_n	Parts of system including panels, cones, surrounds, and domes (frequency, Q, and amplitude)
Settling Resonance	W_{CC1}, W_{CC2}, through W_{CCn}	Node Modes: standing waves, bell modes, delay-coupled interference (weighted notch frequency, Q and amplitude)
Coupling Factors	$\|A\|_1$, $\|A\|_1 \ldots \|A\|_n$	Resonant magnitude (always adjustable parts of resonators +/- dB)
Loss Factors	Q_0, $Q_1 \ldots Q_n$	Resonant bandwidth (default or adjustment possible +/- dB)
Tilt	dB	Departure from frequency-balanced output: B_L factors. Subjective balance and default placement equalization
Wave Interference	Td	Dimensions: enclosure, floor, wall, speaker and listener placements.
Equalization	EQ	User operated: room, other traditional
Power	Pw	Maximum low-frequency input or output. Analog proportional or threshold switch detector.
Mode Switch	SW1, SW2, etc.	Dynamic process: operate or select

Somewhere between 8 and 32 parameter adjustments might be used to program the compensation system and operate an adequate compensation process. Since

parameters are related to specialized physical and behavioral aspects, data requirements for adjustment ranges and precision are much smaller than needed for coverage of human perception. Eight-bit resolution is adequate for many of these, so that a one-time data stream of 1-5 kbits characterize a very complex response from the model. A description of the foregoing parameters follows.

Low Crossover (L_X) or Acoustic Coupling – Low frequency output depends on the size of radiating surfaces. The 6dB-per-octave loss from this relationship is inherently compensated by increased cone motion provided the back voltage from the motor structure is small compared to the driving signal. If compliance were infinite and the motor produced force without velocity restriction, this idealized configuration would have flat response. Real systems having air volumes, mechanical stiffness, and velocity limits have two practical crossover points where the response breaks from flat and converges to 6dB/octave and then to 12dB/octave bass loss. Usually, the second inflection is near the lowest useful response of the loudspeaker and often becomes impractical to compensate. The control parameter for this compensation is frequency in Hz and the conjugate response (specific compensation) is a +6dB/octave boost for decreasing frequency. This boost starts at the first response inflection from the speaker. The value for this parameter can be measured by applying a test signal, *e. g.*, sine wave, to the speaker and measuring the response. The value may also be calculated based on physical characteristics of the speaker such as cone and coil mass. The value may also be derived from a speaker model such as a standard Theil/Small model which is generally provided by the speaker manufacturer.

High Crossover (H_X) or Mass-Compliance Factor – High frequency output depends on the size and velocity of the radiating surfaces. Usually, the wavelengths of interest for frequencies near cutoff are small compared to the motional part of the speaker creating the output. Stiffness and damping properties of the cone material affect this transition from whole surface radiation at lower frequencies. Higher frequencies radiate nearer the voice coil. Moving mass eventually creates a 6dB/octave reduction of motion with increasing frequency. Leakage inductance from the motor assembly adds further compliance to the system to create a second inflection making a 12dB/octave drop at the highest practical frequencies. These losses, combined with decreased radiating area, create a reduction of acoustic output. At lower frequencies, the two relationships can be skillfully balanced to create flat acoustic response. The control parameter for this compensation is frequency in Hz,

and the conjugate response is a 6dB/octave boost with increasing frequency. The value for this parameter can be measured or calculated in a manner similar to that previously described for low crossover.

Low Boost Shelf (L_S) – Maximum boost from L_S is restricted to practical maximum limits from the amplifier power, cone motion, voice coil length, enclosure size, and intended bass extension. These requirements conflict and interact, *e. g.*, longer voice coils require bigger amplifiers and suspension parts. For example, an extended coil can provide greater linear excursion and bass output potential, but efficiency decreases unless flux energy from the magnet is increased. Increased excursion requires bigger surrounds and the cone diameter must be increased to maintain the same radiating area and bass output. Bass resonance is lower, but the speaker must be physically larger, and the increased moving mass further reduces efficiency. If amplifier power is limited the enclosure becomes larger. Manipulation of the L_X, W_0 (described below) and L_S parameters makes efficient or extended response loudspeaker designs with much less compromise. In addition, the L_S parameter can be made to change or track different volume control settings and program dynamics, so that the speaker system can operate near its maximum capability for a wide range of conditions. The L_S parameter is affected by design and construction factors, which include radiation area, compliance, and force factor of the driving motor. The modeled parameter can track changes to these factors with very little interaction. The L_S parameter is +/- dB. The L_S parameter setting can be determined by experimenting with compromise tradeoffs via test measurements.

High Boost Shelf (H_S) – Practical maximum performance and power handling capability of the speaker limit the maximum high frequency boost. In some applications the H_S parameter might be set to reduce perceived distortion from a program source instead of from the speaker. As with L_S, the internal operational settings can be made to change by command. This feature could help cover up increasing distortion when other parts of the system, including program material, is pushed to operate above their maximum linear power capability. The H_S parameter is +/- dB. Its setting is best determined experimentally via test measurements.

Low Peaking (W_L) – A resonant bass cutoff is often preferred. Recordings are balanced for this and most speakers are designed to have self-resonant bass enhancement. Generally, larger speakers have lower resonant frequencies and listeners associate value to this relationship. The W_L parameter provides a low

frequency resonant boost below the natural self-resonance (W_0) of the speaker itself. When a low Q_L compensating response dip removes the W_0 effect, the speaker system behaves electrically and sonically as if it were larger. It will not have the boom or box sound of an equalized small speaker. To prevent excess distortion and power draw, the signals below W_L are attenuated rapidly and the high-pass filter involved tracks the W_L parameter setting. This action has subjective factors similar to weighted compensation, since frequencies below cut-off can be weak but still audible when not masked by other sounds in the W_L critical band of human hearing. Generally, W_L peaking is adjusted and set to control power and distortion for maximum playback conditions. For other conditions, the high-pass or sub-bass filter can be shut off and parts of the low-pass filter (L_P) moved or changed without obvious subjective consequence. As mentioned, the W_L parameter can be dynamically modulated to extend bass or reduce distortion. Adjustments to the response of the low-pass filter include peak frequency (W_L) and amplitude $|A_L|$ in dB. Default Q_L setting is implied.

High Peaking (W_H) – A resonant cutoff may be preferred to compensate for a second cutoff inflection, or it can be used to restrict reproduced bandwidth. The first application can extend response and reduce group delay distortion. Bandwidth restriction might be needed to make a distorted program sound better. A peaking resonator and tracking high-pass filter (H_P) operate in a similar manner to the low-frequency peaking system. The high-pass filter can be aligned to create a weighted response limiting and to provide a peaked cutoff response preferred in contemporary design practice. High-pass filter adjustments include peak frequency (W_H) in Hz and amplitude ($|A_H|$) in dB. Default Q_H setting is implied.

Motor (B_L) – Properties of the magnet structure and voice coil are consolidated to a traditional representation. Voice coil diameter, winding length, gap flux, pole dimensions, etc., are parts of traditional speaker models creating motor and generator equivalents. Force and back EMF relationship, with electrical current and mechanical motion, as well as other factors related to mechanical properties, get simplified representation. Sometimes leakage inductance and mechanical mass relationships are included. These elements are measured, calculated, or derived by various means to create a group of motor-related parameters that can affect many performance aspects of the speaker. Adjustable parameters to the compensation system and its processing are dependent on B_L. Therefore, if needed, the model can have relationships set up so that B_L changes can modulate parameter settings. For example, the B_L specification

can be used to track or readjust other parameters for similar speakers having different magnet weights, voice coil lengths, or other electro-mechanical factors. The adjustable B_L parameters can be: flux x length, volts x velocity, peak excursion length, and resistance/inductance or time constant.

Bass Resonance (W_0) or Self-Resonance – This is the natural bass resonant frequency of the loudspeaker. Its value is determined by air volume, mechanical compliance, radiating area, damping, moving mass, motor characteristics, and other design features of the speaker. Unlike traditional bass models, an acoustic coupling factor is not part of the W_0 response, since it has already been accounted for in the L_X parameter of the compensation system. This arrangement prevents interaction between adjustment parameters and the equivalence to mechanical behavior transforms to an LCR resonator equivalent of mechanical behavior. User command language of frequency W_0, Q_0 and coupled energy or amplitude $|A|_0$ can specify conjugate responses to the mechanical resonant behavior. Because the speaker behavior and correction responses for the system are of the same order, but opposite amplitude, the group delay performance can be very good. Adjustments are W_0 in Hz, Q_0 as a number, and $|A|_0$ in dB. These adjustment values can be experimentally determined by very-close-range instrumentation, position sense transducers, or by calculation.

Port Tuning (W_B) – Many speakers are designed to provide a secondary bass resonance intended to extend frequency response or improve power handling. Physical structures to do this include ports, tubes, passive resonators, labyrinths, other woofers, etc. Much literature has been devoted to modeling and tuning these systems, usually to achieve flattest possible frequency response. Such models, combined with the compensation system, have the potential to extend bass, improve transient response, and improve efficiency without adding cost to the speaker. Greater power handling and extended bass capability can be structurally designed and the resulting frequency response irregularity and end band distortion compromise from using this strategy can be removed by conjugate correction. When this choice is made, ports can be larger and tuned lower, enclosures made smaller, along with other changes to improve bass. The resulting consequences of rough response and sub-bass overload are reversed by the compensation system. Flags or switches can be part of a control operation, which identifies program conditions or response states at the loudspeaker. Limit conditions from samples or tests determine switch states for this feed-

forward/feed-back type system. Then, parameter coefficients are chosen from the switch or status inputs. In this manner, the compensation system can operate with fuzzy logic controls or other means to prevent unnecessary toggling or changes to parameter values. For example, different volume control settings, program levels, or bass content can change parameter coefficients and alignment to achieve a higher power or flatter response optimization. The switching operation is likely to be audible, but the control hysteresis provides strategic changes that are less apparent or objectionable. Inexpensive equipment is intended and likely to overload or be pushed beyond a reasonable linear operation. The compensation system operating with appropriate control logic can activate soft and loud operational states, each having optimum alignments, without creating a continuum of distortions inbetween.

Mechanical Resonance (W_1, W_2, through W_n) – Many parts and actions of a speaker system behave like resonators. Unlike W_0, most of these are unintentional and they usually add to the acoustic output of the speaker. These include flexure and mass behavior of the domes, cones, and surrounds, as well as cabinet resonance, among others. Equivalent LCR behavior is parameterized to non-interacting individual corrections or adjustments similar to those for W_0. Resonance adjustments are W_1, W_2, through W_n in Hz, Q_1, Q_2, through Q_n (described below) in units, and $|A|_1$, $|A|_2$ through $|A|_n$ (described below) in +/- dB. Often a small, low-Q resonance that appears similar to room responses has sonic impact. Unlike a test microphone used for traditional response leveling, a listener can move around and, in time, can sense and become aware of signal decays that collapse to broadband resonance. Conjugate correction eliminates this difficulty and helps to improve the outcome from ancillary equalization systems.

Settling Resonance (W_{CC1}, W_{CC2} through W_{CCn}) – The hidden compensation processes using multi-resonator notches and weighted side energy enhancements are placed in frequency and adjusted by these control parameters. Sonic consequences from delayed nodal resonance and standing wave interference is diminished or removed.

Coupling and Loss Factors ($|A|_0$, $|A|_1$, through $|A|_n$ and Q_1, Q_2, through Q_n) – Q parameters for some frequencies can be estimated default values which work for materials, designs, and construction materials common to speakers. Since it is almost impossible to casually determine Q from a response plot or simple measurement, the

fixed value is practical, and the default choice simplifies and reduces the number of adjustments.

Wave Interference (Td) – The finite dimensions of speaker enclosures can create wave-related interference pressure behind the radiating surface. Usually this problem is ignored because of the complexity of the correcting response. Expensive speakers are built with big motor assemblies having tight magnetic coupling to combat back-wave interference. Often these systems have inner baffles, layerings of different types of absorbent material, specially shaped rear cavities, or transmission line tubes to help remove the interference. Typical speakers exhibit constructive/destructive wave interference from their simple box type enclosures, which ultimately cause response ripples of increasing occurrence with frequency. The irregularity worsens as the shorted-turn effect, and damping from B_L, are reduced by smaller magnets, reduced steel and other economic factors. Corrections are made by a conjugate opposing response from a delay-and-feedback-loop process. Although an analog implementation is possible, DSP methods are more economical and by their nature make good equivalence to the physical behavior and problem to be fixed. A similar correction can reduce reflected interference from the floor, table top, walls, etc.

Tilt (dB) – Most contemporary listeners prefer a speaker whose response drops slightly with increasing frequency. When a speaker is made less expensive its response tends to rise. The compensation system corrects many problems and changes that occur from reducing magnet weight, improving efficiency and extending high frequency response. However, the flat acoustic balance created from these conjugate corrections may sound either muddy or thin compared to a contemporary tone standard. Tilt creates a compensating response slope. The tilt parameter operates like a tone control and is used to achieve a desired bass-treble balance. Internal manipulation of the L_X and H_X, as well as traditional response leveling methods or processes create this response.

Delay or All-Pass - This parameter moves the signal in time to compensate for transducers radiating high and low frequencies from different places. Adjustments can reduce group delay distortion, improve phase match to a sub woofer and align transient response. Adjustments can be made using physical or electrical measurements.

FIGS. 8a through 8f show control panel displays and menus used to adjust parameters. Measurement data, design expressions, and response curves provide useful information to adjust or tune the system to the loudspeaker. The system generates conjugate responses to those of the speaker and silently removes delayed or long-settling mechanical sound. Experience has shown better performance for many listening positions compared to more complex conventional leveling practice. Common digital processes can be utilized for the implementation of these functions. Exemplary known digital processes are shown in Appendix A. Alternative known digital implementations can be used to achieve a wholly similar result.

The compensation system may also be implemented using analog circuits. However, parts of multi-resonators or coupled functions are not likely to track wide adjustment ranges unless they are complex. Hence the analog circuits are shown for their functionality and similarity to well-known better DSP equivalents. These circuits have adjustable parametric controls, but they require component changes to simulate the parametric alignment accuracy inherent with the digital system. The following examples are traditional circuits that have semi-parametric adjustment capability and produce conjugate responses like those described.

Cutoff Filters – Sallen and Key circuits can make active high-pass and low-pass filters with adjustable peaking responses. Figure 9 shows the two filters configured to a single op amp. Mid-band response is flat and cutoff slopes are + and – 18dB per octave. Peaking amplitude of L_P and H_P are adjustable and correspond to FIG. 4b. However, the frequencies must be preset for this circuit. For this example, the two sections do not interact with each other because of their wide frequency separation.

Constant Slope Equalizers – RC time constants are changed in an active feedback circuit to create response curves like FIG. 4a. The circuit shown in FIG. 10 is the boosting half of a sophisticated parametric tone control. Its adjustment range is limited for good parametric independence between shelf and crossover.

Parametric Notches – Figures 11 and 12 show simple analog notch circuits having a wide tuning range. Notch depth stays constant but Q increases with frequency setting. Component values can be chosen to provide a reasonable approximation to Q_0, Q_1 through Q_n for practical adjustment ranges.

Weighted Compensating Notches – A composite filter element is made up from one or more sharp notches, each having energy added to either or both sides of

the reject frequency. When its rejection and boosting energies and bandwidths are carefully chosen, the filter system can remove unwanted energy with little compromise or alteration to sonic balance. Reject notches, for this purpose are sharp, have high attenuation and are generally high-Q. The side band compensations or restorations are best made from boost responses at both sides of the rejection. However, a one-sided compensation may work better for speakers having combined response roll off and interference energy problems. The unsymmetrical boost helps flatten the frequency response. Other aspects are the same. Compensations can be very small, have low Q, and the average pink noise energy in a one-half to one octave band centered to the correction is constant when the process is on or off.

Decay from sounds can perceptually shift to low-Q side-band frequencies. Consequently, a double-tuned alignment for better immediate transient response settling is preferable to a one-sided or single-tuned boost method. The resulting composite filter can be carefully tuned and scaled so that it can be silently tuned over a useful frequency range. Without weighting, the small response losses on either side of the correction contribute to a nasal sound. The coloration may be subtle but when more corrections are used the losses can overcome any advantages. This is particularly true if the response is just made flat, as it might with standard-practice equalization. Weighting eliminates the compromise and allows multiple corrections to be more effective and free of sonic interaction and also eliminates loss when unintentional correction is applied to a speaker having different behavioral properties.

For either implementation, neutral weighted response notches can be made up from resonant and anti-resonant responses added to the signal. With reference to FIG. 13, a correction element might incorporate a single high-Q notch, whose frequency is centered on the behavior mode. One low-Q boost response is placed at the same frequency to provide the compensating equal-weight energy. The circuit portion of FIG. 13 shows a practical combination of an active circuit notch and passive LRC boost resonator in a feedback path. Both parts create the weighted response notch. However, the tuning range is limited, and the low-Q boost is likely to be audible, since the human hearing perception can resolve transient sounds decaying to a low-Q resonance.

A better alternative is to use two low-Q resonant boost responses, each tuned and placed at one or both sides of the high-Q notch. Figure 14 shows an example made from four tunable bi-quad resonators or state variable filters. Two of the filters

create close spaced high-Q notches and the other two provide the boosted low-Q energy at the outer high and low sides. These state variable filters are easily created with DSP processors, and they maintain constant Q and notch depth over a wide tuning range. The analog system creating the responses shown in FIG. 15 has eight precision variable resistors ganged on a single shaft to tune the W_{CC1} or weighted notch parameter. Figure 13 shows a circuit for one of four sections. Figure 15 shows response curves for two different Q settings and these correlate to those in FIG. 4d. The double-tuned boosts can be aligned to yield a faster and smaller settling response to transients. A double-tuned notch like that from the figures offers similar advantages and also provides a dead-band or band-reject capability to accommodate manufacturing tolerances from one speaker to another. Side frequency boost is still needed and double-tuned resonators are best used.

Other variations and simplifications can be made when adjacent behavioral modes have similar properties, as is likely with most speakers. Two or more rejection notches can share Q and amplitude settings as well as compensation boost. Combinations include two notches with three boosts, two notches with two asymmetrical boosts, three with two, etc. A single low-Q boost with a frequency halfway between two notches can be used. Three low-Q boosts with frequencies below, above, and between are a better variation. For all of these implementations, the notch depth is often great and the side frequency boost is usually small. Usually, the overall energy response to random noise averaged about the compensation region is made to be the same or slightly higher than without correction.

Delayed Interference or All-Pass – A hybrid analog-digital CCD device can create a small, convenient tunable delay. Though performance may be poor, they can be connected or configured like the example in FIG. 16 to provide interference-like behavior. The circuit can create approximate conjugate responses to wavelength related reflection and transmission behavior from walls, tables and the insides of speaker enclosures or other parts of the transmission path or system. The circuit can be set to create an inverted comb filter or additive interference like response which would be opposite in time, phase and amplitude to subtractive interference loss from reflecting surfaces. The correction boosts where interference takes away. The circuit can also be adjusted to have a comb filter like response to cancel additive energy from reflections within the speaker enclosure. Better time delay interference filters or comb filter like responses can be made from DSP processes. Both the analog and the DSP

can be configured to be relevant to the physical reflection model and like other parts of the correction system, are controlled by parametric adjustments related to physical behavior. The delay interference path filter has controls relating to dimensions, surface absorption, and the amount of interference correction needed.

With reference to FIG. 16, Td relates to the difference between the direct path from the speaker to listener and the longer bounce path also from speaker to listener. Td also relates to the out and return path between the speaker and an opposite surface inside an enclosure. A wall behind the speaker can be characterized the same way. Larger Td gives a larger distance. RC relates to surface roughness or absorption at high frequencies. Larger RC product for greater loss or faster attenuation of upper frequency comb filter response and correction. The control R1 adjusts the magnitude of the response or correction. CW direction increases subtractive responses while the CCW position near the + input to the op amp gives maximum additive responses. The circuit produces an interference response whose amplitude decreases with frequency. This matches or simulates losses of absorption materials of practical speakers. Much of the irregular response from small speakers can be experimentally changed to something that appears to be more easily processed by the compensation system. Usually, the delay setting to do this matches the back arrival wave relationship expected from the speaker enclosure. When it does, this one adjustable parameter equals a multitude of conventional response-leveling processes.

Figure 17 is an all pass or phase shift network. Its frequency response is flat however its output is in phase and high frequencies and out of phase at low frequencies. The circuit alters transient response without changing frequency response. The variable control increases the transition frequency as it is turned CW. This element is useful to correct group delay and other transient related responses.

As previously mentioned, mechanical disturbances produced by speakers are often audible, yet invisible or hard to interpret from response measurements made using a frequency sweep and microphone. Traditional compensation methods using a deep notch usually result in either a nasal sound or undesirable balance aberration. The weighted compensated notch filter of the present invention solves this problem and yields some other advantages as well. When two drivers (woofer and tweeter) are crossed over by just a capacitor or by circuits that overlap frequencies, one or both drivers can have interference compensation without perceptual loss to the other. The

same applies to different listening positions. One position having a bad response can be compensated without compromising the sound for other listening positions. The correction is hidden by the weighted side energy. Since good listening positions are not compromised by the correction, a wide range of listening positions can have good sonics. This feature is particularly useful to horn-type loudspeakers for theater sound, where slightly different corrections are needed from one installation to another. By using default optimization for a class of speakers or construction features, parametric control and adjustment is simple and intuitive.

Some of the most important behaviors of loudspeakers (with respect to acoustically perceptible effect) cannot be modeled or implemented from traditional methods. Such behaviors include interference and resonant coupling, as well as nonlinear consequences from such behaviors. In addition, other spatial factors related to stiffness of moving parts and high frequency de-coupling for motions away from a driving voice coil need to be considered. Any of these can create source movements, delayed energy release and phase error to binaural hearing. Often, such destructive responses can be invisible or very difficult to interpret from traditional microphone-and-spectrum-analyzer calibration methods.

An example of a parametrically addressed compensation for a specific physical effect is compensation for mechanical de-coupling in large full-range speakers. Such speakers are usually designed to have the entire cone move at low frequencies. At high frequencies, only the inner part of the driver is the primary active radiator. The rest of the cone is intentionally de-coupled to attenuate its nodal breakup. This design choice changes the position of an equivalent radiation source. A linear correction to move high-frequency radiation forward in time uses complex pass filters to create band-limited delay of the lower frequencies. Then, as frequency increases, a latent delay decreases thereby maintaining a phase match to the response order of the speaker. Some group delay distortions can be removed this way. A physical dimension from the speaker along with attenuation and speed-of-sound properties of the cone can yield information to specify a correction from a dedicated filter process. However, the delay effect and de-coupling frequencies can be experimentally determined to yield parameter values. If some of the speaker parts change size or attenuation properties, new values can be extrapolated.

An additional example of a parametrically addressed compensation, which is difficult or impossible to compensate using traditional methods, is compensation for

Doppler modulation in small speakers. Computer and multi-channel sound systems require small speakers to play very loud. The resulting combination of fast, high-displacement cone motions can impart an additive or subtractive velocity to the sound reproduced. The low-frequency movements of the speaker cone impart a Doppler modulation effect by moving the effective high-frequency radiating area. The effect is quite annoying and creates a sound typical of small speakers struggling to play loud. When radiating area, frequency, and acoustic power output are known, a reasonably accurate calculation of the resulting cone motion and Doppler radiation factor can be made (see FIG. 5). An opposing alternating time delay is created for correcting the loudspeaker response. This multiplicative or non-linear process can make the equivalent radiation source steadier so that non-linear distortion is reduced. A good behavioral model of the speaker and a single dimension parameter can specify the correction. Any such open-loop or feed-forward non-linear correction is very difficult to accomplish. Modeled time-delay modulation is much better as it provides the exact conjugate response correction. The example is included here because cone motion from bass drivers is reasonably predictable from the model, and instantaneous correction from DSP methods will become more practical in time. Experiments were performed by using CMOS-type analog delays operating from voltage-controlled oscillator clocks. One path compensates for latency from both "bucket brigade" devices and the other receives the variable clock to produce delay modulation.

For nodal flextures, wave interference and non-linear motion situations, the removal or attenuation of behavioral responses by using the weighted-notch process is a good compromise. The weighted compensation can be parametrically moved to where it is needed. If needed, its severity can be modulated to best match program dynamics, distortion from the loudspeaker, and human critical-band hearing perception.

Additional bass output capability and response extension can be had when the dynamics of the audio signal are detected and used to change parameters or hidden compensations. This technique can work for any speaker system operating from the improved method.

However, if parts of the speaker are designed to require the dynamic compensation, additional performance features are possible. An adjustment list or bit map intended for the specialized system can have the same parameters and adjustments as the general-purpose system, *i. e.*, the controls have features in

common. Thus an electronic system with a compensation system is capable of receiving instructions to compensate and optimize a variety of speaker models and products which can be attached to it, both generic speakers and those designed specifically for use with the compensation system. When needed, the system can improve clarity, imaging, and bass extension without changing sonic balance, *i. e.*, the sonic signature of a product line is maintained.

The adjustment effect that parameters create are described or represented by a symbol or acronym as included in the previously described table. In this way, mathematical statements, circuits, loudspeaker design features and other specialized language are consolidated to lists that a programmer or user can intuitively interpret. Then, response-curves, behavioral features, wavelength relationships, as well as other things important to human perception, become relevant to the process model used for design and/or compensation of a loudspeaker system. Compensation is simplified since interactions between design and performance changes and basic relationships are meaningful and intuitive.

Settling and nodal compensation parameter adjustments include frequency in Hz, Q as a number, and rejection in dB. Since multiple-order nodal behaviors are possible, the Q and rejection adjustments are more likely to have similar values for more than one compensation frequency. Controls can be linked so that two or more frequencies in Hz can specify corrections for nodal behaviors sharing a mechanical or interference property in common.

Bell-mode compensation for speaker cones provides a good example. Several methods can determine these frequencies. Multiple nodes or wave-related interference can be observed using sine wave excitation and optical interferometry. Waterfall plots from MLSSA-type stimulus, as shown in FIG. 7, or a differential microphone probe can be interpreted to reveal settling energy from breakup. These behaviors are audible and the hidden compensation process can be manually or slowly swept in frequency to reveal sonic changes during pink-noise reproduction. The experimental method requires caution and experience since room reverberation can imitate the mechanical sounds. Correlation between the three techniques is good, and when several frequencies from similar causes are determined, composite weighted notches of similar Q and shared boost parameters simplify the correction.

The above parameters and their adjustments change a configured compensation for a specialized behavior of the loudspeaker. Some processes are inherently confined

to a narrow range of frequencies where a particular kind of behavior is most likely to occur from the loudspeaker system. Others affecting time and response slopes cover a wide spectrum. These dedicated operations are isolated enough from each other that interaction between them is small. Selected parameters can be changed without affecting others. In addition, the data requirements are modest for useful adjustment resolution.

As described in the foregoing speaker example, controls adjusting the compensation system are related to physical parts and acoustic properties of the speaker. Other speakers of similar, but not exact design or construction can be improved in the same manner. The control or specification information to do so has the same simple, intuitive language. A small inexpensive speaker could be substantially improved with 16 parameter adjustments and a few default settings. To match the complex response from this system, a conventional multi-band or graphic equalizer would need well over 100 band, and much more control information.

The compensation system uses slopes, crossovers, and other mathematical functions to create predicted corrections for characteristics of the speaker. These operations are controlled by coefficients or points, and by process commands instead of from documenting and calculating responses to linear plots. Since each process has a specialized nature, the range of frequencies, amplitudes, Qs, etc., confine parameter ranges to small portions of a control space. A three-word code can create an accurately placed resonator or complex weighted response notch subsystem. In the case of 8-bit code words, this map or information sequence gives 128 points for a logarithmic frequency decade, +/- 20dB in 1/2dB steps for amplitude, and 16 Q settings, as well as space for process commands and control formatting. For this example, 384 points or frequencies in the audio band would be available for setting sharp rejection notches and weighted compensations. Bitmaps for other parametric controls and process commands may require less resolution, thus providing even greater economy. This compact specification feature could be used for bar-code installation of the compensation system to correct a specific speaker design or for dynamic control of processes from codes hidden in data streams such as used in HDCD systems. As described in detail below, portable devices, such as hearing aids and communication systems benefit when the scope of processed information is reduced.

For many applications, 384 points or frequencies in the audio band might be chosen for a successful process incorporating ten basic operations, *i. e.*, shelves, notches, compensations, cutoff filters, delays, *etc.* Process scale increases very rapidly with resolution: rejection notches can be sharp and need accurate tuning, interference compensations are inherently complex and speaker responses from a traditional measurement practice reveal little useful information to simplify processing. Unless the compensation system is designed to recognize and process sub-system or modeled responses from curves, the corrections must be done on a point-by-point basis. Even a good response correction done this way requires substantial process power. The scope of conventional processing widens when time, phase, and settling issues are considered. For this situation the number of filters or processes needed to resolve a useful correction response anywhere in a three-dimensional grid (such as a waterfall plot) can be impractical. Some economy is possible when filters or processing power are allocated to those portions of the grid requiring higher resolution and more correction activity. Even these efforts fail to achieve the hidden or compensated correction benefits of the compensation system.

A good example of this difficulty is evident from FIG. 6. The compensation system has 8 notches, one weighted notch for node correction, and a low frequency boost crossover and shelf. The high frequency drop provides a crossover to a tweeter. Clearly, the difficulty of plotting points and creating a response synthesis is evident. The compensation system creates this conjugate curve from 24 adjustments (L_X, L_S, L_P, W_0, Q_0, $|A|_0$, W_1 through W_7, $|A|_1$ through $|A|_7$, W_{CC1}, and one default Q setting).

A strategy similar to compensating the removal of in-band signals can be used for the highest and lowest frequencies. For example, distortion caused by excess cone motion from bass energy can be less objectionable by reducing the input to the speaker. One way to do this is to restrict the low frequency response to a practical value. Quite likely, the speaker would have been able to make feeble bass reproduction below the cutoff, provided the program material does not mask the low bass signals. Wide dynamic range program material can force the speaker to produce perceivable output at these frequencies. Unfortunately, the resulting feeble output is frequently covered up by other sounds in the same perceptual band of human hearing. Hence, most of the time an extended response from the speaker is useless and only adds distortion and consumes amplifier power.

The weighted response correction of the compensation system can be applied to the audible side of either a low or a high frequency broadband or cutoff rejection or filter. If done carefully, the perceptual response appears unaltered and can be made to sound extended as it was before filtering. Large systems intended to reproduce high-power, sub-audible or infrasonic energy require a different approach, although the same method applies.

The compensation system combines a loudspeaker model and conjugate process to create a zero-phase filter whose acoustic output has low group delay distortion. For this application, the peaked response not only complies with the standard practice but is used in the alignment to help improve group delay and becomes part of a hidden compensation that allows the band limiting feature to be turned on or off.

When neutrality like that described above is chosen, the compensation system can be made to change its cutoff depth and compensating energy with minimum audibility. This feature allows the compensation system to track or modulate from changes of volume control settings, power levels, or program conditions competing for perceptual bands. In this manner, the bandwidth restriction can force more power to the speaker before distortion sets in. Since audible energy is now rationed a smaller amplifier can produce the same playback volume.

These factors have economic importance. Speakers can have parts designed so that the system can benefit from the compensation system. They would have shorter voice coils and lighter-weight cones to produce higher acoustic output from an amplifier. Efficiency is higher, but mechanical behavior is often compromised and the acoustic output from such a design tends to rise with frequency. Without correction the speaker sounds harsh and shrill and therefore must be compensated. However, once these changes are corrected by the compensation system, the speaker plays louder, can be smaller, and a lower-powered amplifier can be used.

If virtual bass extension is chosen, the peaked or compensated response is moved in frequency with a carefully worked out strategy. Process settings and the related acoustic peaking frequency at a band edge becomes dependant on bass content of the program material, power reserve of the amplifier, and motion capability of the speaker. When playback level is high, and low frequency energy is strong, the peaking response is moved up in frequency to maintain a reasonable cone motion and distortion performance.

Most entertainment systems have equalizers, tone controls, and other user-operated or initiated features. Acoustic visibility is needed. The compensation system works best in tandem with these and can improve their performance. Once the speaker system has the more idealized radiation behavior provided by the compensation system, various response leveling systems will not have to sort out or guess which parts of a measured response taken from a microphone originate from the speaker or from the room. The compensation system can more effectively rule out inappropriate equalizations from these response leveling programs which might destroy phase response or be inappropriate for other listener positions. Tandem operation also assures removal of distracting mechanical sounds caused by responses that are unidentifiable by traditional methods. Human binaural perception is good at perceiving, recognizing, and localizing these sounds. Once they are removed, the traditional equalization methods can boost high frequencies without added harshness, and low frequencies without box boom or muddy sounds characteristic of minimal construction.

For the compensation system, specialized analog functional block circuits can make simple practical systems. Response ranges for their adjustable parts can be predicted from the conjugate model so that parallel signal paths and multiple shared functions are possible. Much fewer active devices are needed in a signal path, thereby reducing costs and distortion. Some time-related correction is possible from all-pass filters but other delay operations are impractical.

Exemplary analog and DSP implementations of the compensation system in a loudspeaker context are provided in appendices B and C, respectively. Descriptions of other applications of the compensation system follow.

Communication System - Communication systems can be subject to unwanted narrow-band noise or tones at fixed or varying frequency. Quite often some time latency is available allowing a subsystem to identify and track such tones. When it is not practical or possible to create an out-of-phase signal to remove the disturbance the compensation system achieves an effective result. Frequently, it is possible to track the disturbance with the parametrically moveable weighted response notch. Unless an intelligence signal has the same frequency as the one being rejected, the correction will be effective yet not degrade or change the sound. Frequently, communication systems have transducers such as microphones, headphones, and speakers as part of the sound reproduction system. These elements could have undesirable responses that

can be compensated individually, or collectively as a system. One or more processors can be placed anywhere in the signal chain. They can be controlled or programmed to compensate a multitude of component part mixes, such as transducers, signal paths, human hearing requirements, and the like. To set up such a system, control parameter information may be identified or accessed by hidden codes in the signal path. Auxiliary channels, as well as other means, can be used to convey this information. Since the compensation system uses this information efficiently to create very complex responses, the correction process can be installed in many systems. Examples include computer speakers, telephone systems for hearing impaired, and mobile communications. For these systems, the processors can be external or not part of the end user appliance. Hence, the compensation system can be programmed to a particular model of appliance and externally activated. Part, all, or none of the processor can reside with the appliance.

Hearing Aid – Many hearing difficulties arise from damage to hair cells within the auditory nerve. Frequently people suffer tinitus or head ringing, and have difficult understanding conversations in crowded noisy rooms. Sometimes a closely spaced group of cells corresponding to a narrow frequency band has had mechanical damage causing an excessive nerve response to high intensity stimulus. This action often sounds discordant and painful. It may cause disruption to a hearing feedback path to and from the brain, so as to initiate premature protection functions, thereby aggravating the unpleasant sensation. Since the stimulus causing the mechanical motion is in a range of frequencies easily defined by medical test routines, the parametric weighted notch filter can be easily tuned to the same frequencies to block sound transmission that excites these cells. If the additive weighting is set up properly for the individual the perceptual loss to other sounds is reduced. In a typical situation the notch must have a band reject characteristic which may have to become wider with sound pressure level. The compensating side energy tracks these changes to reduce the intelligibility loss to other sounds. A system designed to perform this correction can be more effective and its specialized processing might take up less battery power than general-purpose response contouring methods.

Watermark Identification – A recording can be made with known frequencies removed by a sharp notch. If the same recording needs to be identified later a system can be made to search for the known missing frequencies. One can expect flutter, noise, and unpredictable clock rates to shift dead bands and to create a multitude of

difficulties. A recording may use music signals to define operating frequencies and timing of encoded security notches. Clock dependency is reduced and number keys are used to prevent someone from finding these frequencies and altering their purpose. During playback, key identified parts of the program initiate decoding parameters used for verification. Like the communication system previously described, the recorded notch should be silent or hidden as best as possible and it is likely to jump from one frequency to another. The parametrically controlled weighted compensation method is ideal for this application, and the combination of keyed frequency jumping and hiding provides for an effective security system.

Feedback Elimination from Sound Systems – Large sound reinforcement installations, phones, and other systems involving the simultaneous acquisition and reproduction of sound are subject to feedback. Usually, a microphone picks up some part of a reproduced acoustic signal and adds to the signal to cause oscillation. Sophisticated DSP parts of speaker phones can sort out room echoes and phase relationships of cross talk to clarify sound transmission and eliminate feedback. Frequency shifters randomize feedback phase to prevent a slow buildup of howl or high-Q loose-coupled oscillation. Sometimes, the latency from these methods interferes with stage performance or synchronization. Tunable weighted notches can be set to sensitive frequencies to eliminate oscillation without significant loss to sounds. Double or multiple-tuned versions have little interaction with each other, and the inherent dead band makes tuning out feedback much easier.

CD Player – A typical electronic system, operating in either the digital or analog domain (or both), might combine intentional user adjustments and the compensation system. Parts descriptions, mathematical models, measured data, and human preference information may be combined and incorporated into its design. Once parameters are defined and their coefficient values are determined an organized group of numbers can program an electrical or mathematical system to perform specific correction response for different shapes and sizes of loudspeakers. The parameter list, and even the individual correction modules, can be downloaded to a digital signal processor in a CD player or other device. This operation can be done at any time, so that it provides a programmable upgrade. It can be activated by bar code, remote control, downloaded from a computer, or installed to dedicated player/speaker combinations.

While several particular forms of the invention have been illustrated and described, it will be apparent that various modifications can be made without departing from the spirit and scope of the invention.

WHAT IS CLAIMED IS:

1. An apparatus for modifying an electrical audio signal for input to a sonic reproduction device characterized by a plurality of individual responses which in combination define an overall response for the sonic reproduction device, each individual response comprising at least one of a frequency, time, phase or transient response, said apparatus comprising:

a plurality of modification filters having modification responses that simulate the plurality of individual responses, the modification filters for receiving the electrical audio signal, modifying the electrical audio signal and providing the electrical audio signal to the sonic reproduction device; and

a plurality of adjustable parameters, each associated with at least one of the modification filters for allowing adjustments to the responses of the modification filters;

wherein the adjustments create a plurality of individual conjugate responses, each individual conjugate response associated with at least one of the plurality of individual responses.

2. The apparatus of claim 1 wherein the plurality of individual responses of the sonic reproduction device are related to at least one of mechanical, acoustic and electromagnetic behavior of the sonic reproduction device.

3. The apparatus of claim 1 wherein the filters are defined by digital signal processes.

4. The apparatus of claim 1 wherein the filters are defined by analog circuitry.

5. The apparatus of claim 1 wherein the plurality of modification filters are non-interacting.

6. The apparatus of claim 1 wherein the plurality of modification responses combine to form an overall response that is a conjugate to the overall response for the sonic reproduction device.

7. The apparatus of claim 1 wherein at least one of the modification filters comprises a cut-off filter and the parameters for adjusting the frequency response of the cut-off filter comprise peak frequency, amplitude and Q parameters.

8. The apparatus of claim 7 wherein the peak frequency, amplitude and Q parameters modify the frequency response of the cut-off filter in at least one of the low and high frequency ranges.

9. The apparatus of claim 1 wherein at least one of the modification filters comprises a constant slope equalizer and the parameters for adjusting the frequency response of the constant slope equalizer comprise crossover frequency and boost shelf parameters.

10. The apparatus of claim 9 wherein the crossover frequency and boost shelf parameters modify the frequency response of the constant slope equalizer in at least one of the low and high frequency ranges.

11. The apparatus of claim 1 wherein at least one of the modification filters comprises a parametric notch filter and the parameters for adjusting the frequency response of the parametric notch filter comprise notch frequency, amplitude and Q parameters.

12. The apparatus of claim 1 wherein at least one of the modification filters comprises a parametric notch-boost filter and the parameters for adjusting the frequency response of the parametric notch-boost filter comprise notch frequency, amplitude and Q parameters.

13. A sound compensation system for altering an electrical audio signal for input to a sonic reproduction device having associated behavioral characteristics, said system comprising:

a model of the sonic reproduction device having a plurality of filters that simulate at least one of the behavioral characteristics of the sonic reproduction device, each filter having an associated response that combine to define an overall response

for the model, each response comprising at least one of a frequency, time, phase or transient response; and

a controller that modifies the response of each of the plurality of filters to transform the filter into a conjugate filter having a responses that is a conjugate to the original response of the filter.

14. The system of claim 13 wherein the behavioral characteristics are defined by individual components of the sonic reproduction device

15. The system of claim 13 wherein the behavioral characteristics are defined by groups of individual components of the sonic reproduction device

16. The system of claim 13 wherein the filters are defined by digital signal processes and the controller comprises a computer

17. The system of claim 13 wherein the filters are defined by analog circuits and the controller comprises adjustable circuit components

18. The system of claim 13 wherein the sonic reproduction device comprises a speaker and at least one of the plurality of filters comprises at least one associated adjustable parameter and the value of the parameter is calculated based on physical characteristics of the speaker.

19. The system of claim 18 wherein the physical characteristics of the speaker comprises at least on of cone and coil mass, air volume, mechanical compliance, radiating area, damping, moving mass and motor characteristics.

20. The system of claim 13 wherein the sonic reproduction device comprises a speaker and at least one of the plurality of filters comprises at least one associated adjustable parameter and the value of the parameter is derived from a standard speaker model.

21. The system of claim 13 wherein at least one of the plurality of filters has at least one associated adjustable parameter and the value of the parameter is determined experimentally using standard test measurements.

22. The system of claim 13 wherein the controller is configured such that an adjustment in the setting of one parameter modulates the setting of at least one other parameter.

23. The system of claim 22 wherein the sonic reproduction device comprises a speaker and the one parameter that modulates the at least one other parameter relates to the magnet structure and voice coil of the speaker.

24. The system of claim 13 wherein the controller monitors the program conditions at the sonic reproduction device and sets at least one of the parameter values based on the program conditions.

25. The system of claim 24 wherein the program conditions comprise at least one of volume control settings, program level and bass content.

26. The system of claim 13 wherein one of the filters comprises a weighted compensation notch filter.

27. The system of claim 26 wherein the filter comprises a single-tuned weighted compensation notch.

28. The system of claim 26 wherein the filter comprises a double-tuned weighted compensation notch.

29. A sound system comprising:

a sonic reproduction device having associated mechanical, acoustic and electromagnetic behavioral characteristics;

a source for outputting an electrical audio signal to a model of the sonic reproduction device, the model having a plurality of filters that simulate at least one of the mechanical, acoustic and electromagnetic behavioral characteristics of the sonic

reproduction device, each filter having an associated response comprising at least one of a frequency, time, phase or transient response, the model outputting the electrical audio signal to the sonic reproduction device; and

a controller that modifies the responses of the filters to transform the model into a conjugate model having a plurality of filters with responses that comprise conjugates to the original response of the filter.

30. The system of claim 29 wherein the filters are defined by digital signal processes.

31. The system of claim 29 wherein the filters are defined by analog circuitry.

32. The system of claim 29 wherein the plurality of filters are non-interacting.

33. The system of claim 29 wherein at least one of the filters comprises a cut-off filter and modifications to the frequency response of the cut-off filter comprise adjustments to peak frequency, amplitude and Q.

34. The system of claim 29 wherein at least one of the filters comprises a constant slope equalizer and modifications to the frequency response of the constant slope equalizer comprise adjustments to crossover frequency and boost shelf.

35. The system of claim 29 wherein at least one of the filters comprises a parametric notch filter and modifications to the frequency response of the parametric notch filter comprise adjustments to notch frequency, amplitude and Q.

36. The system of claim 29 wherein at least one of the filters comprises a parametric notch-boost filter and modifications to the frequency response of the parametric notch-boost filter comprise adjustments to notch frequency, amplitude and Q.

37. A method for modifying an electrical audio signal for input to a sonic reproduction device characterized by a plurality of individual responses which in combination define an overall response for the sonic reproduction device, each individual response comprising at least one of a frequency, time, phase or transient response, said method comprising the steps of:

simulating the plurality of individual responses with a plurality of filters;

adjusting the responses of the plurality of filters such that, for each filter, the adjusted response comprises a response that is a conjugate to one of the individual responses;

inputting the electrical audio signal to the filters.

38. The method of claim 37 wherein the plurality of individual responses of the sonic reproduction device are related to at least one of a mechanical, acoustic and electromagnetic behavior of the sonic reproduction device.

39. The method of claim 37 wherein the plurality of filters are non-interacting.

40. The method of claim 37 wherein the plurality of adjusted responses combine to form an overall response that is a conjugate to the overall response for the sonic reproduction device.

41. The method of claim 37 wherein at least one of the filters comprises a cut-off filter and the step of adjusting the frequency response of the cut-off filter comprises the step of setting at least one of peak frequency, amplitude and Q.

42. The method of claim 37 wherein at least one of the filters comprises a constant slope equalizer and the step of adjusting the frequency response of the constant slope equalizer comprises the step of setting at least one of crossover frequency and boost shelf.

43. The method of claim 37 wherein at least one of the filters comprises a parametric notch filter and the step of adjusting the frequency response of the

parametric notch filter comprises the step of setting at least one of notch frequency, amplitude and Q.

44. The method of claim 37 wherein at least one of the filters comprises a parametric notch-boost filter and the step of adjusting the frequency response of the parametric notch-boost filter comprises the step of setting at least one of notch frequency, amplitude and Q.

45. A method of altering an electrical audio signal for input to a sonic reproduction device having associated behavioral characteristics, said method comprising the steps of:

simulating at least one of the behavioral characteristics of the sonic reproduction device with a plurality of filters, each filter having an associated response comprising at least one of a frequency, time, phase or transient response; and

for each of the filters, modifying the response of the filter to transform the filter into a conjugate filter having a response that comprises a conjugate to the original response of the filter.

46. The method of claim 45 wherein the behavioral characteristics are defined by individual components of the sonic reproduction device.

47. The method of claim 45 wherein the behavioral characteristics are defined by groups of individual components of the sonic reproduction device.

48. The method of claim 45 wherein the sonic reproduction device comprises a speaker and at least one of the plurality of filters has at least one associated adjustable parameter and the step of modifying the response of the filter comprises the steps of:

calculating the value of the at least one adjustable parameter value based on the physical characteristics of the speaker; and

setting the parameter to the calculated value.

49. The method of claim 48 wherein the physical characteristics of the speaker comprises at least on of cone and coil mass, air volume, mechanical compliance, radiating area, damping, moving mass and motor characteristics.

50. The method of claim 45 wherein the sonic reproduction device comprises a speaker and at least one of the plurality of filters has at least one associated adjustable parameter and the step of modifying the response of the filter comprises the steps of:

deriving the at least one adjustable parameter from a standard speaker model; and

setting the parameter to the derived value.

51. The method of claim 45 wherein at least one of the plurality of filters has at least one associated adjustable parameter and the step of modifying the response of the filter comprises the steps of:

determining the at least one adjustable parameter experimentally using standard test measurements; and

setting the parameter to the determined value.

52. The method of claim 48, 50 or 51 further comprising the step of modulating the setting of at least one parameter in response to the setting of another parameter.

53. The method of claim 48, 50 or 51 further comprising the steps of:

monitoring at least one program condition at the sonic reproduction device; and

setting at least one of the parameter values based on the at least one program condition.

54. The method of claim 53 wherein the program conditions comprise at least one of volume control settings, program level and bass content.

ABSTRACT OF THE DISCLOSURE

A sound compensation system alters an electrical audio signal for input to a sonic reproduction device having associated behavioral characteristics. The behavioral characteristics of the device are defined by individual or groups of individual components of the sonic reproduction device and include mechanical, acoustic and electromagnetic behaviors. The model includes a plurality of filters that simulate at least one of the behavioral characteristics of the sonic reproduction device. The filters are defined by digital signal processes or by analog circuits and are characterized by one or more of an associated frequency, time, phase and transient response. These responses combine to define an overall response for the model. The filters include adjustable parameters which are used to alter filter responses to produce responses that are conjugates to the responses of the unaltered filters and thus the sonic reproduction device. A controller modifies the parameters.

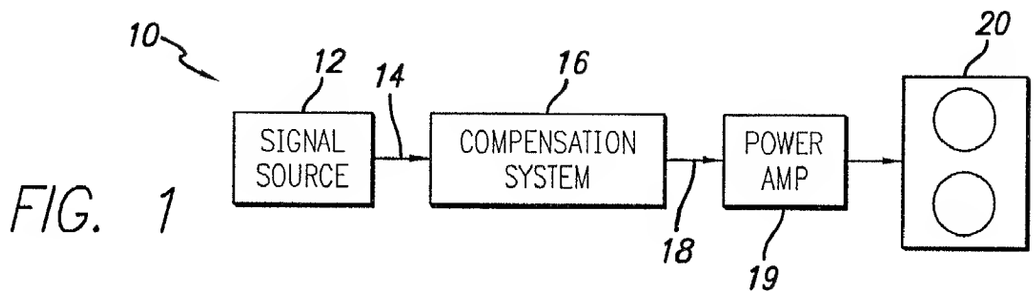


FIG. 1

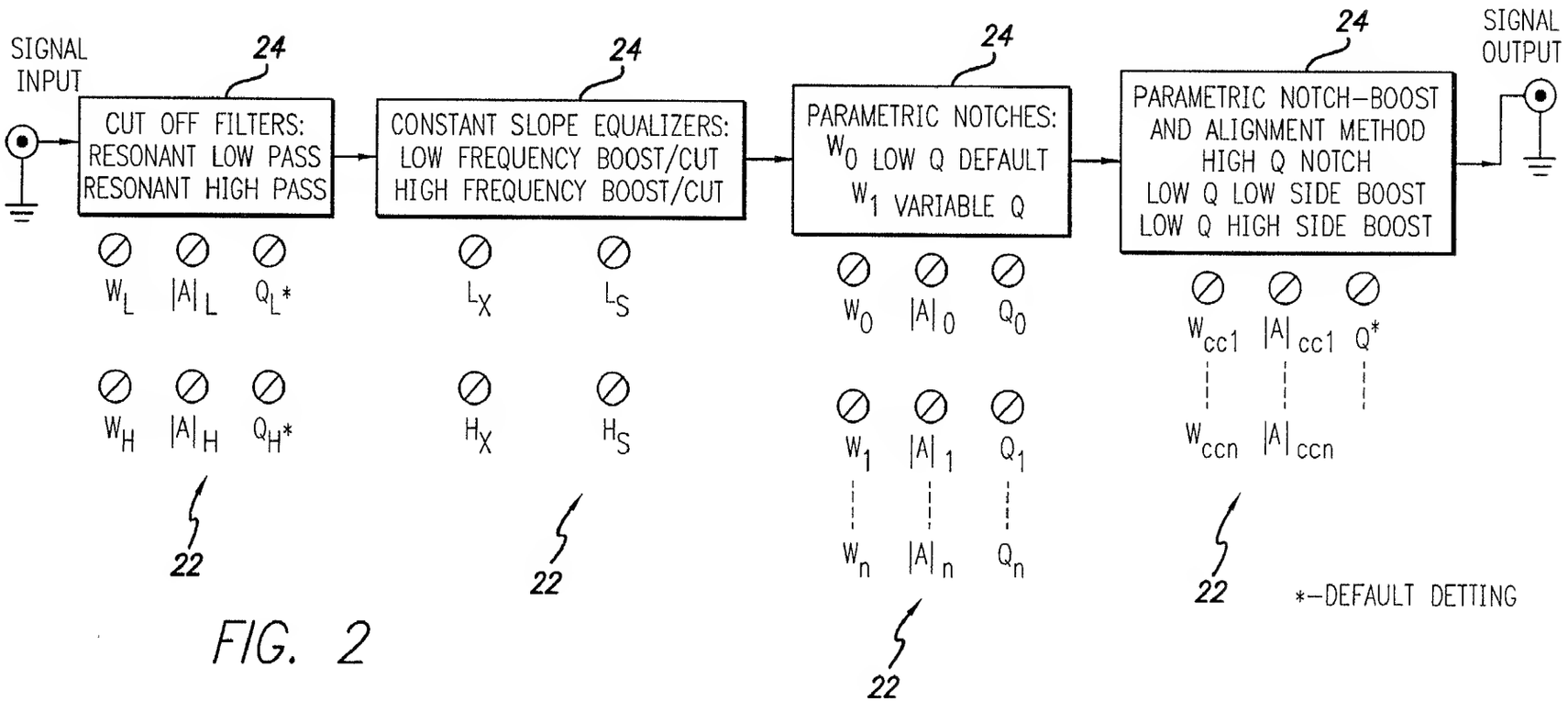


FIG. 2

FIG. 3a

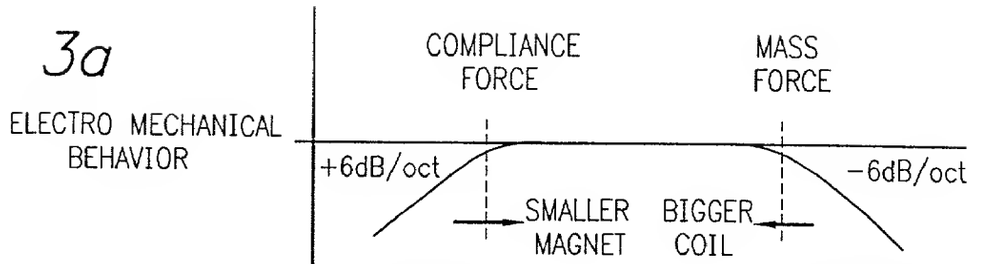


FIG. 3b

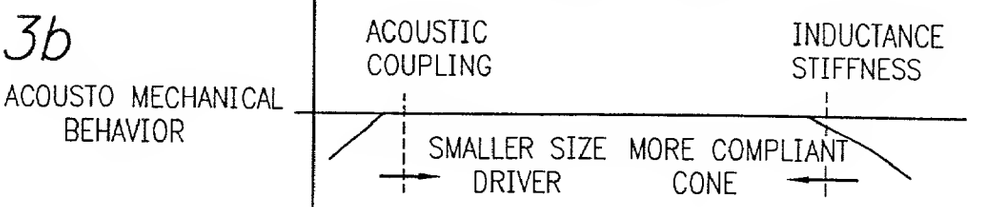


FIG. 3c

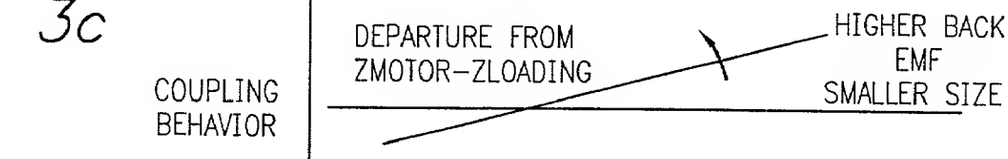
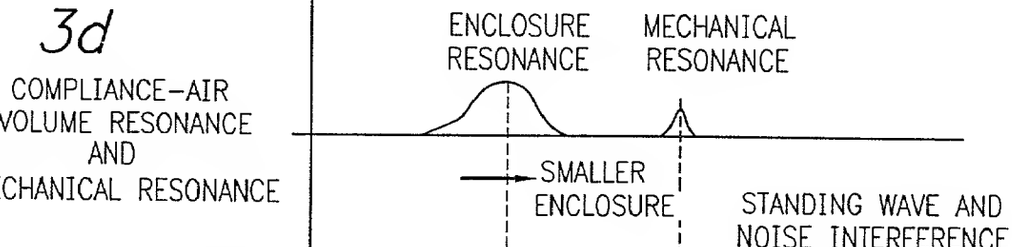


FIG. 3d



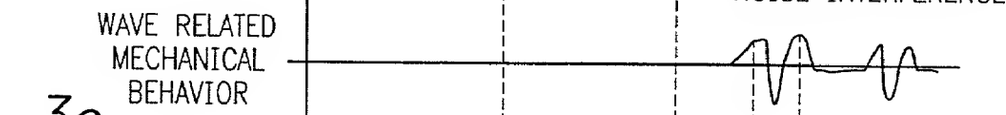


FIG. 3e

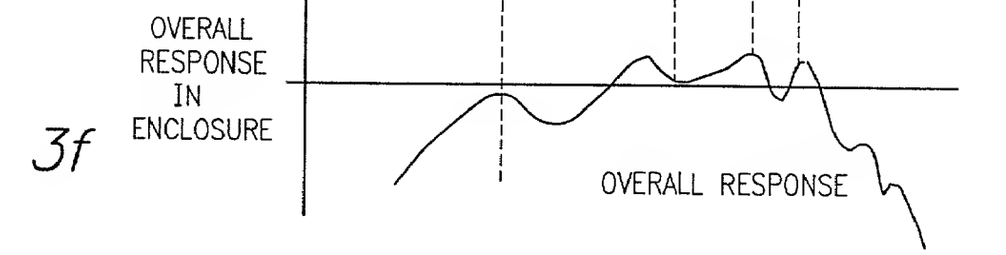


FIG. 3f

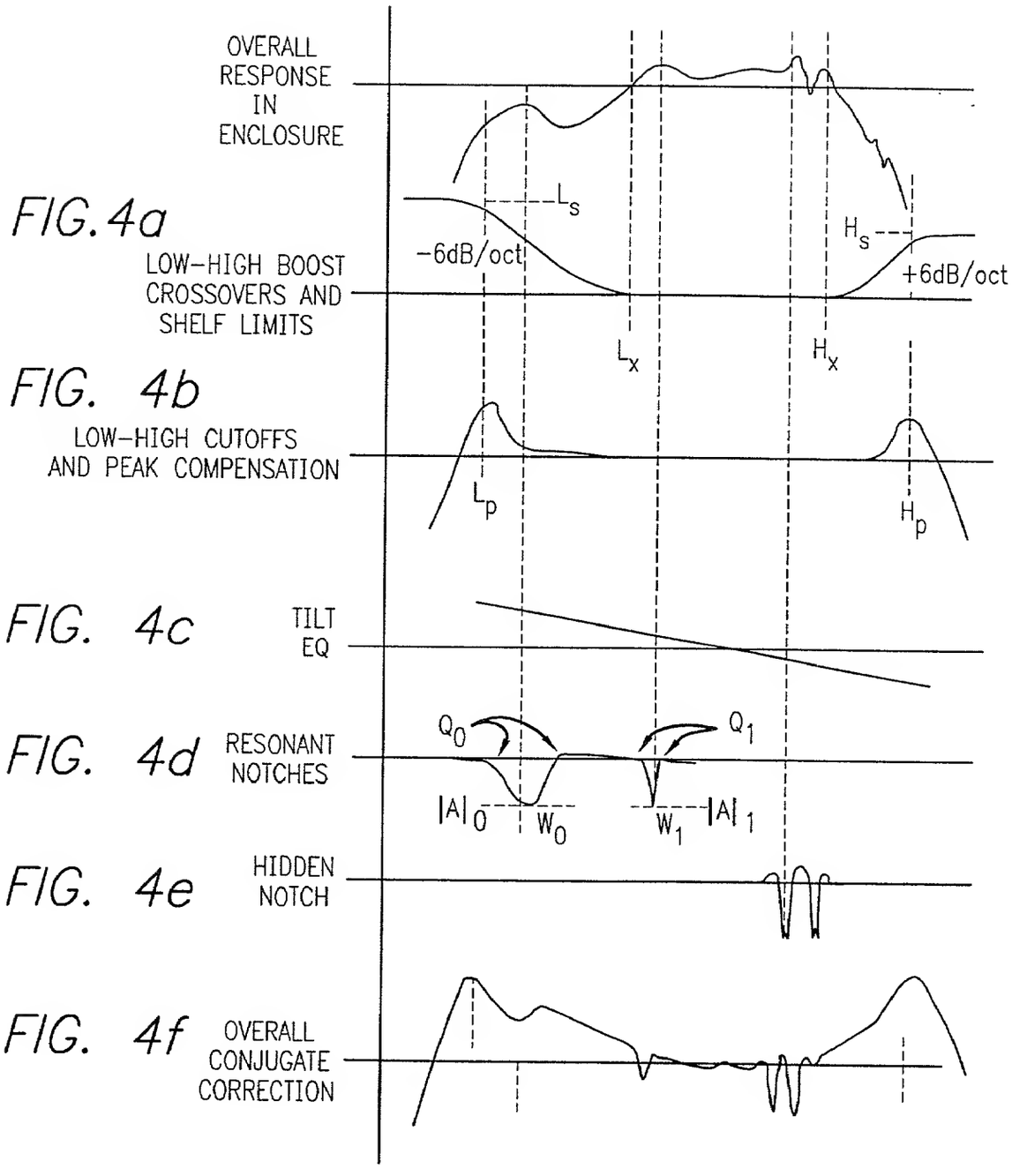


FIG. 4a

FIG. 4b

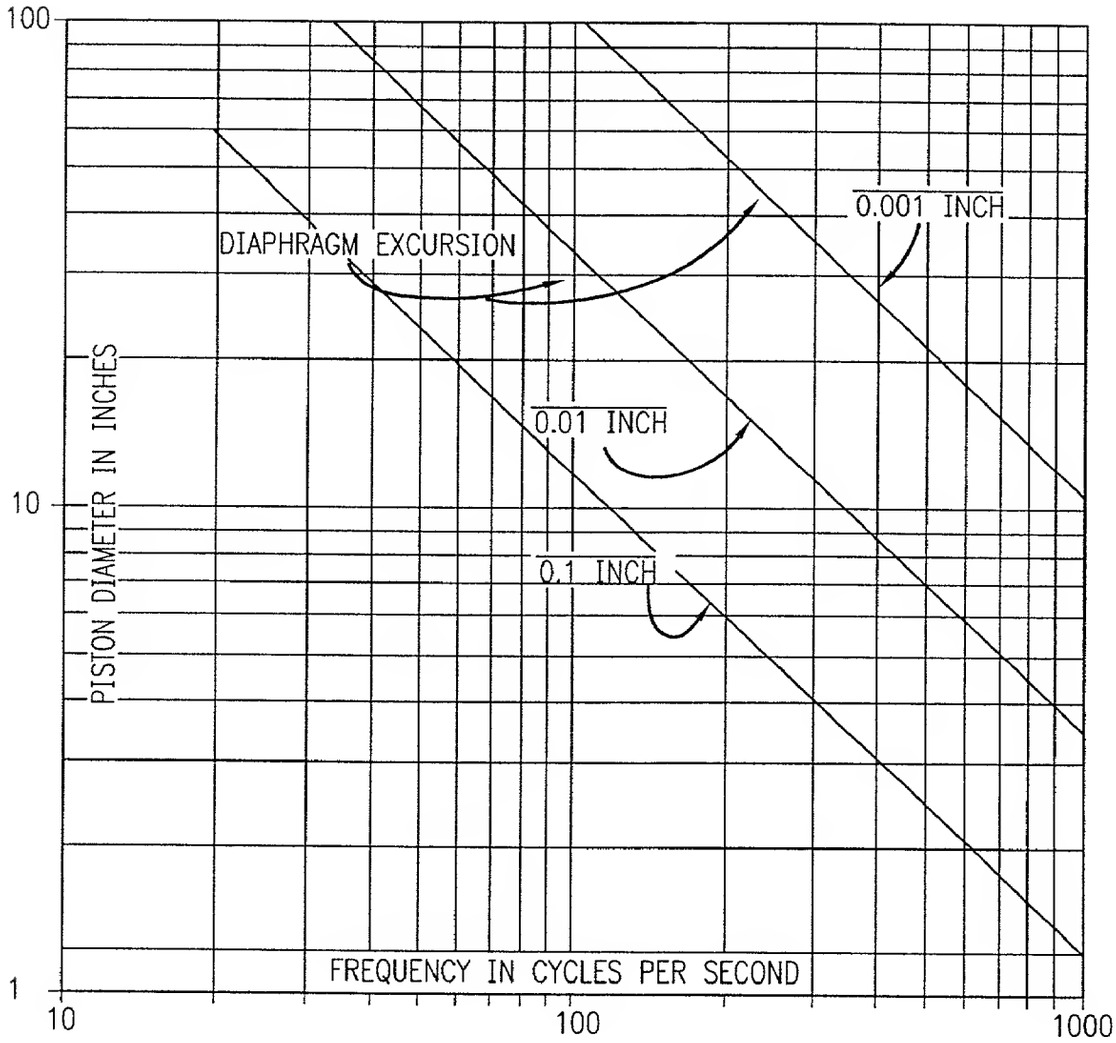
FIG. 4c

FIG. 4d

FIG. 4e

FIG. 4f

FIG. 5


100
10
1
PISTON DIAMETER IN INCHES
DIAPHRAGM EXCURSION
0.001 INCH
0.01 INCH
0.1 INCH
FREQUENCY IN CYCLES PER SECOND
10
100
1000

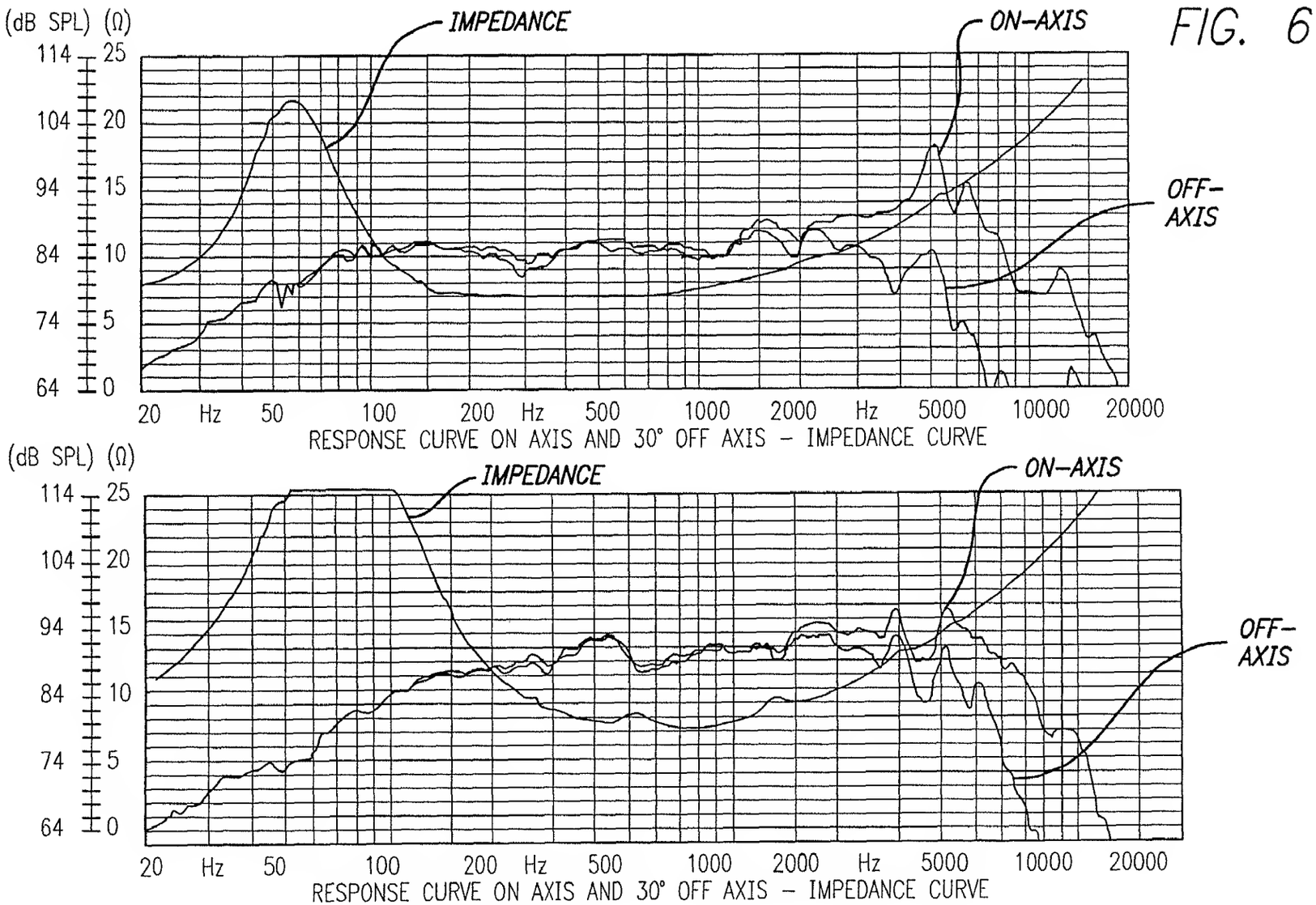


FIG. 6

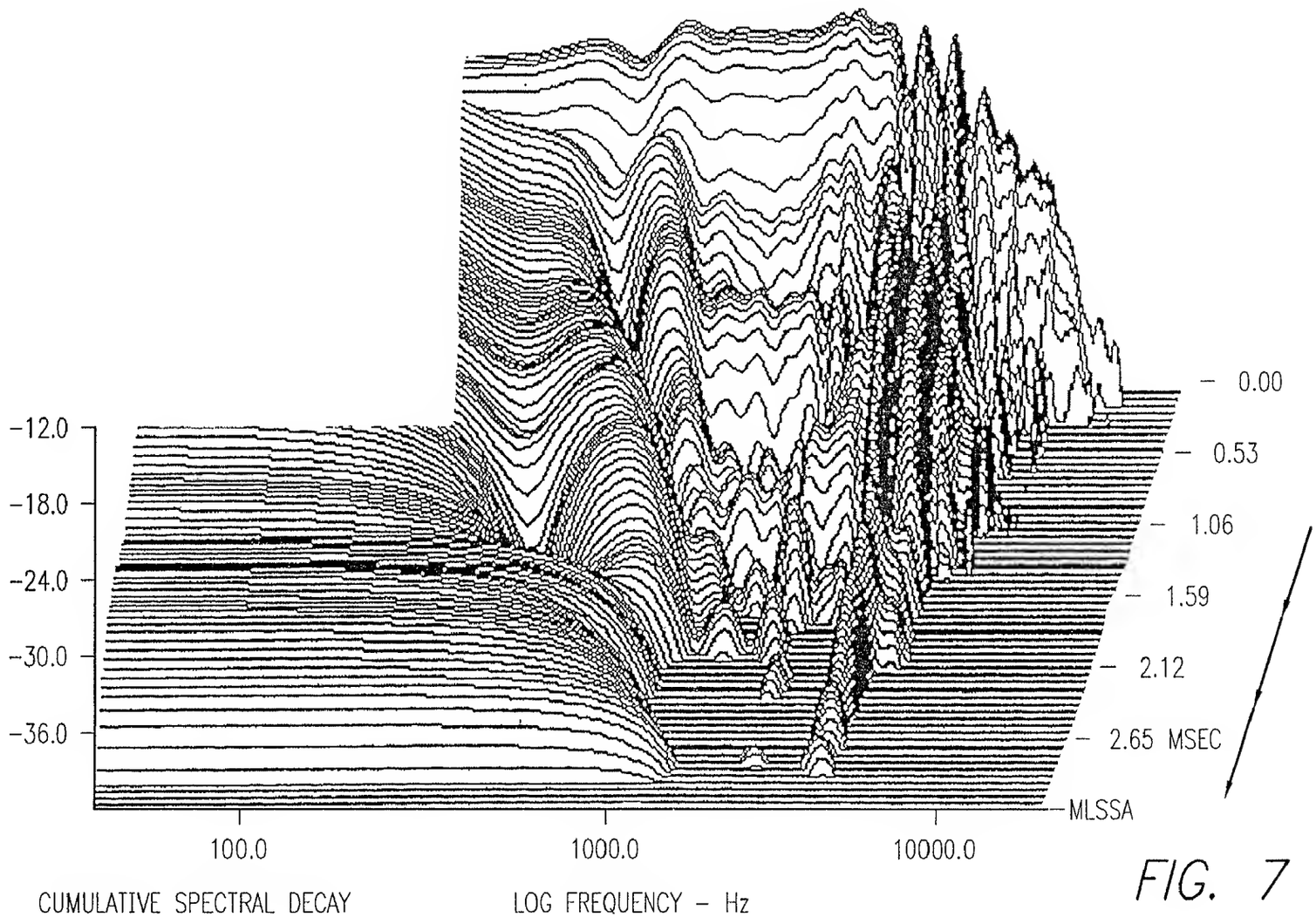


FIG. 7

FIG. 8a

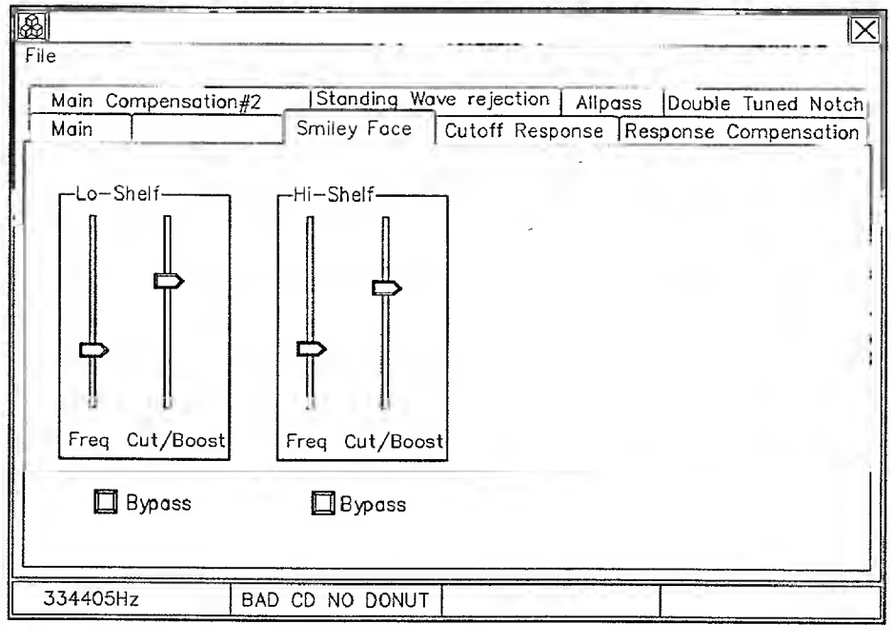

File
Main Compensation#2
Standing Wave rejection
Allpass
Double Tuned Notch
Main
Smiley Face
Cutoff Response
Response Compensation
Lo-Shelf
Freq
Cut/Boost
Hi-Shelf
Freq
Cut/Boost
Bypass
Bypass
334405Hz
BAD CD NO DONUT

FIG. 8b

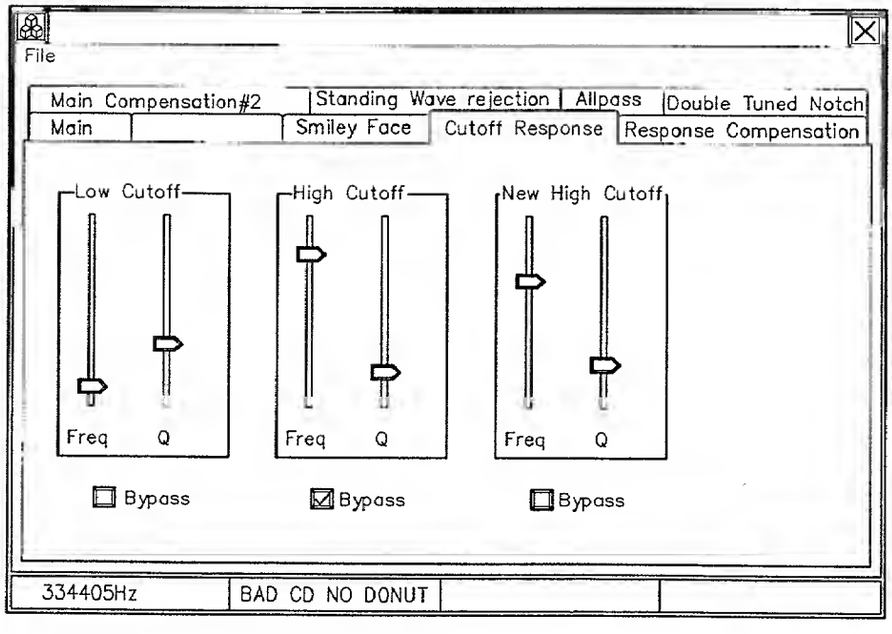

File
Main Compensation#2
Standing Wave rejection
Allpass
Double Tuned Notch
Main
Smiley Face
Cutoff Response
Response Compensation
Low Cutoff
Freq
Q
High Cutoff
Freq
Q
New High Cutoff
Freq
Q
Bypass
Bypass
Bypass
334405Hz
BAD CD NO DONUT

FIG. 8c

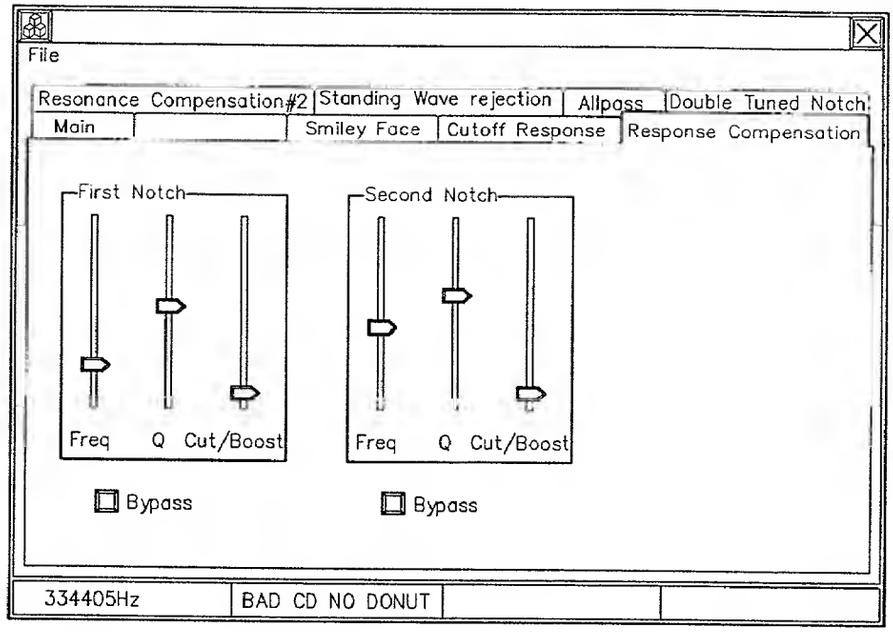

File
Resonance Compensation#2
Standing Wave rejection
Allpass
Double Tuned Notch
Main
Smiley Face
Cutoff Response
Response Compensation
First Notch
Freq
Q
Cut/Boost
Second Notch
Freq
Q
Cut/Boost
Bypass
Bypass
334405Hz
BAD CD NO DONUT

FIG. 8d

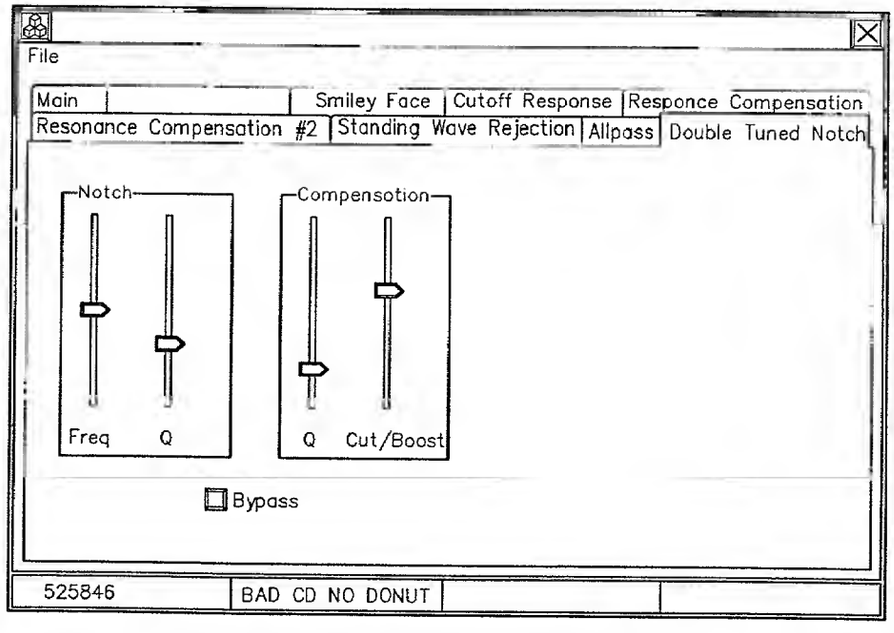

File
Main
Smiley Face
Cutoff Response
Responce Compensation
Resonance Compensation #2
Standing Wave Rejection
Allpass
Double Tuned Notch
Notch
Freq
Q
Compensation
Q
Cut/Boost
Bypass
525846
BAD CD NO DONUT

FIG. 8e

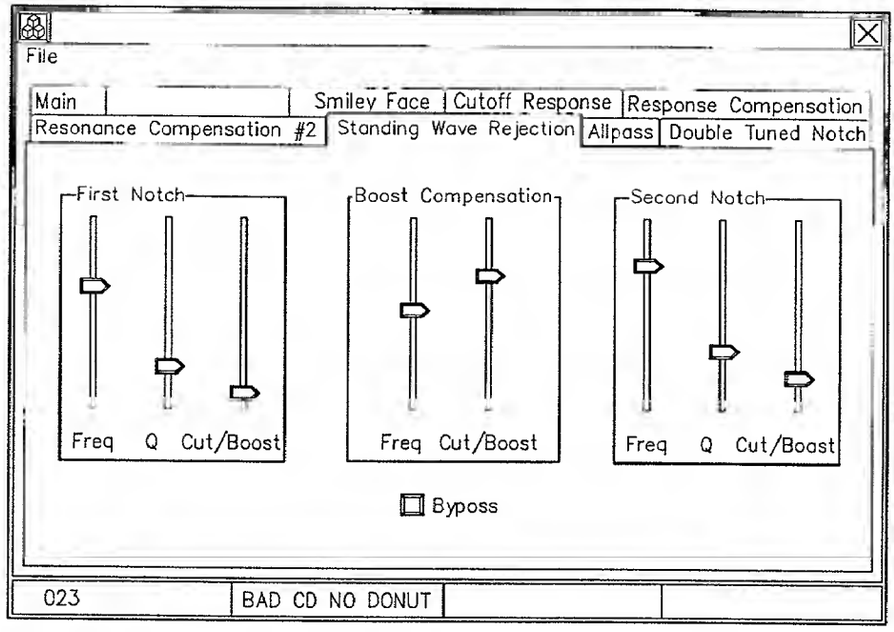

File
Main
Smiley Face
Cutoff Response
Response Compensation
Resonance Compensation #2
Standing Wave Rejection
Allpass
Double Tuned Notch
First Notch
Freq
Q
Cut/Boost
Boost Compensation
Freq
Cut/Boost
Second Notch
Freq
Q
Cut/Boost
Byposs
023
BAD CD NO DONUT

FIG. 8f

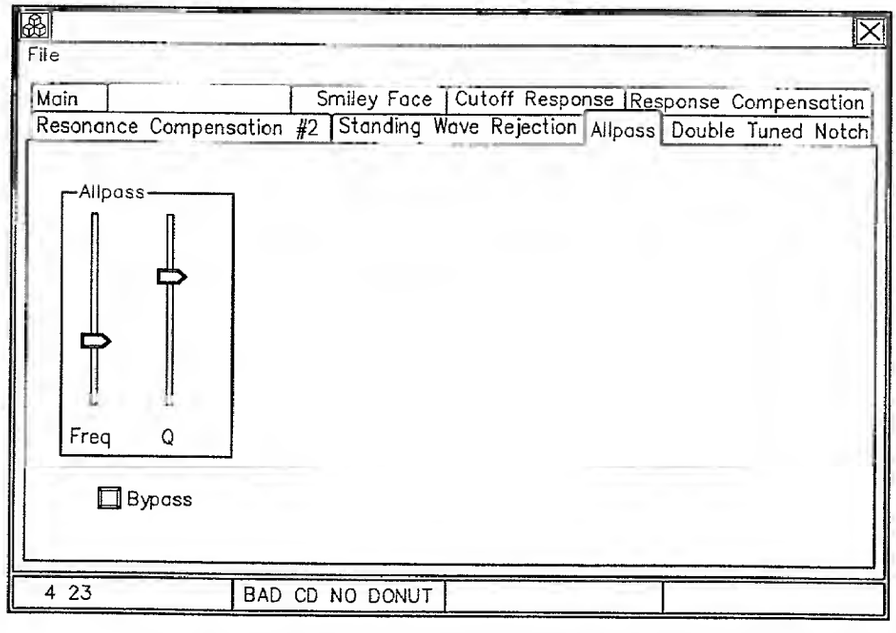

File
Main
Smiley Face
Cutoff Response
Response Compensation
Resonance Compensation #2
Standing Wave Rejection
Allpass
Double Tuned Notch
Allpass
Freq
Q
Bypass
4 23
BAD CD NO DONUT

FIG. 9

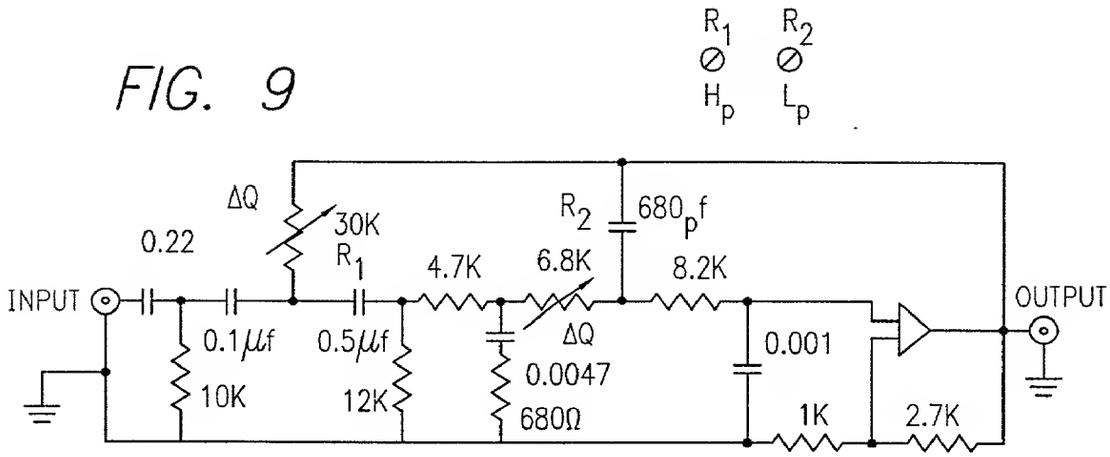

R1
R2
Hp
Lp
ΔQ
30K
R1
0.22
R2
680pf
4.7K
6.8K
8.2K
INPUT
OUTPUT
0.1μf
0.5μf
ΔQ
0.001
0.0047
10K
12K
680Ω
1K
2.7K

FIG. 10

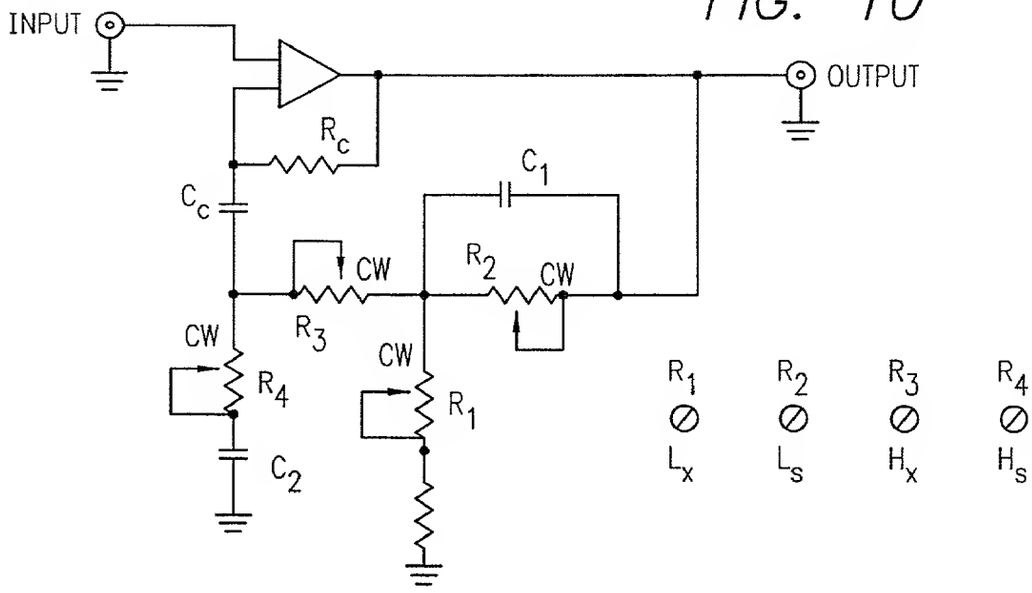

INPUT
OUTPUT
Rc
C1
Cc
CW
R2
CW
CW
R3
CW
R4
R1
C2
R1
R2
R3
R4
Lx
Ls
Hx
Hs

FIG. 11

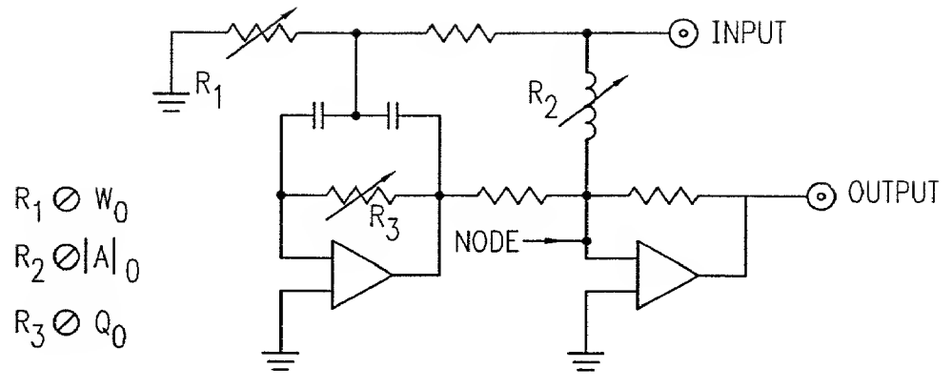
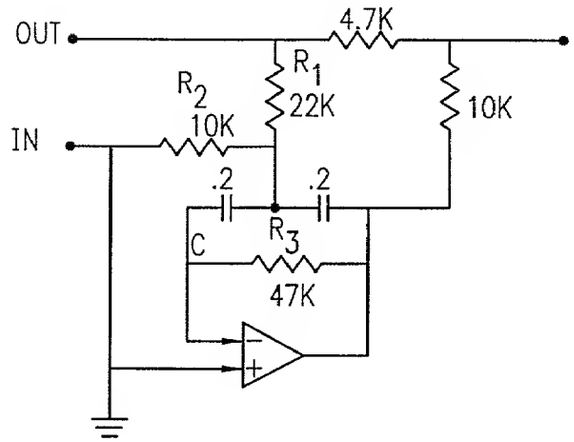
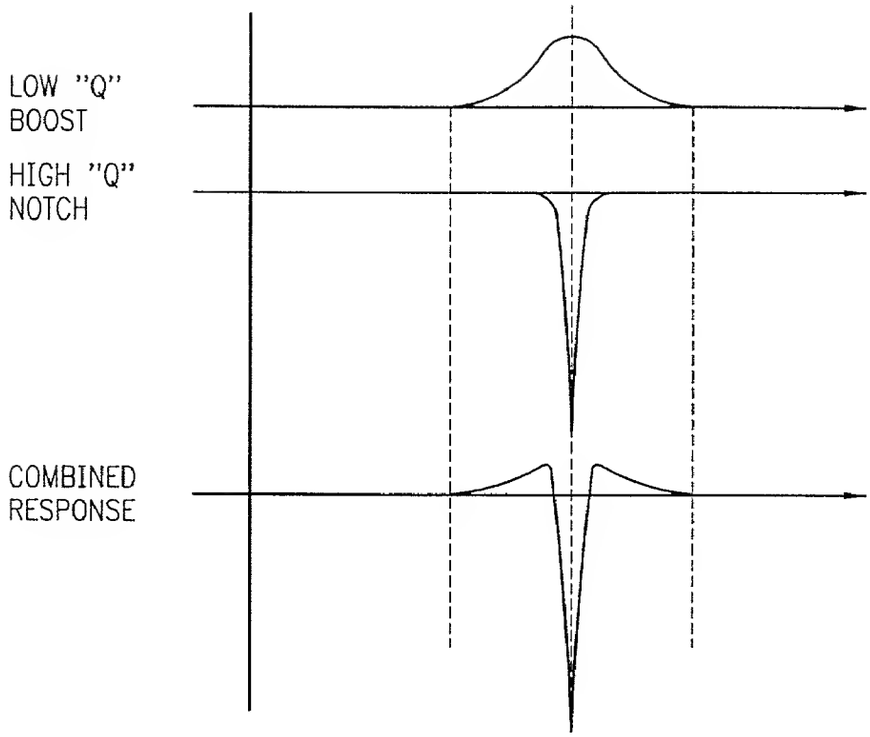

INPUT
R1
R2
R3
OUTPUT
NODE
R1 ⊘ W0
R2 ⊘ |A| 0
R3 ⊘ Q0

FIG. 12


OUT
4.7K
R1
22K
R2
10K
IN
10K
.2
.2
R3
C
47K


LOW "Q"
BOOST
HIGH "Q"
NOTCH
COMBINED
RESPONSE

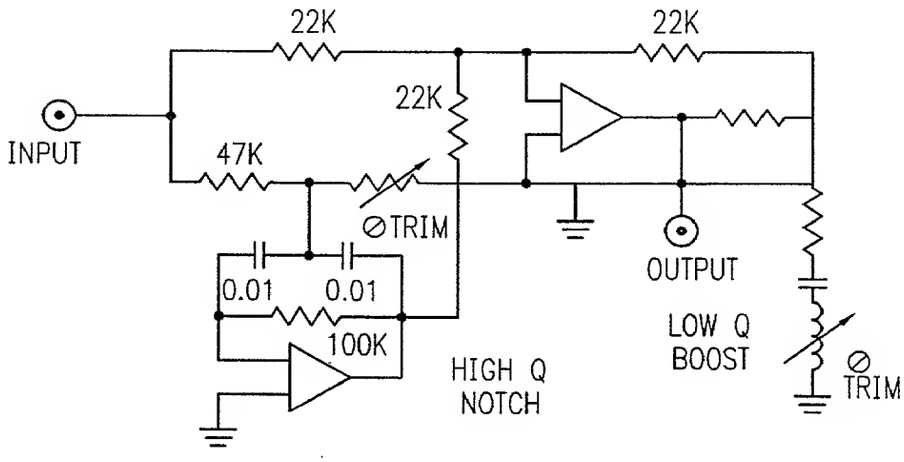
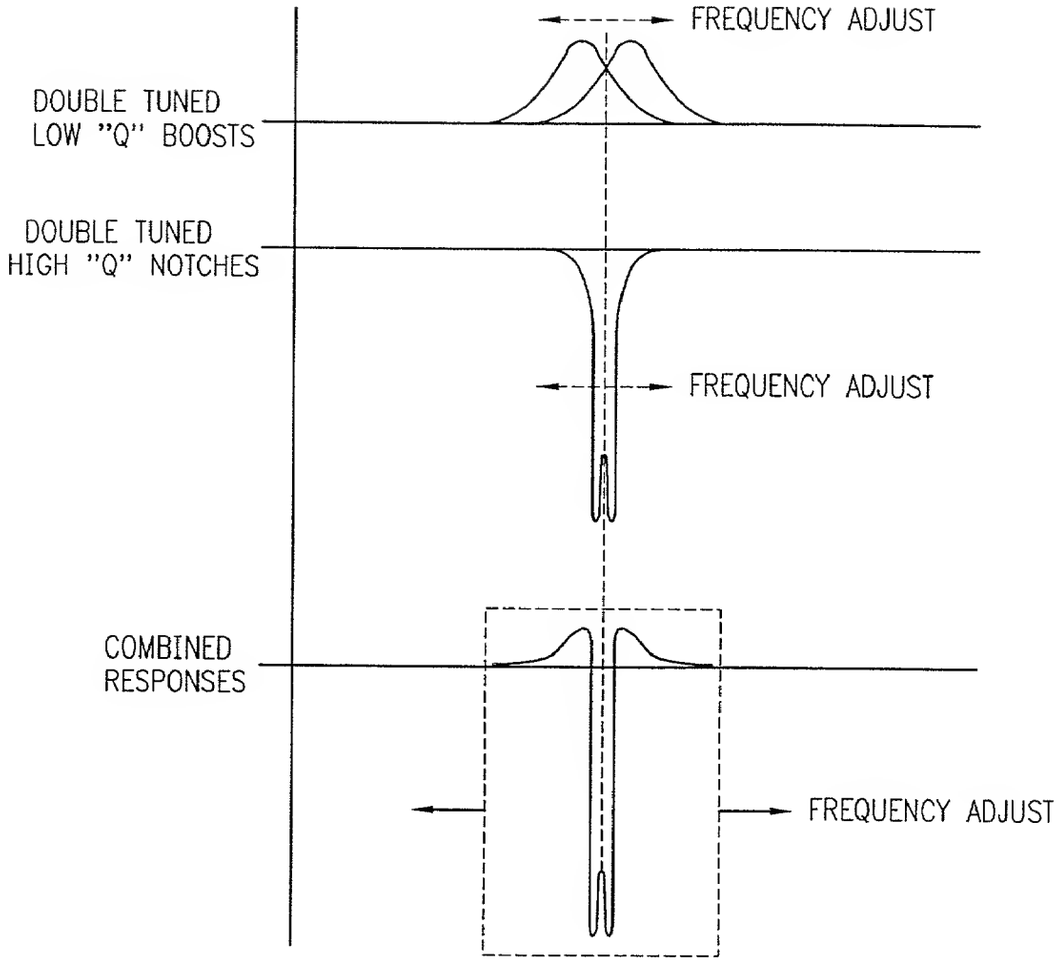

22K
22K
22K
INPUT
47K
TRIM
0.01
0.01
100K
HIGH Q
NOTCH
OUTPUT
LOW Q
BOOST
TRIM

FIG. 13


FREQUENCY ADJUST
DOUBLE TUNED
LOW "Q" BOOSTS
DOUBLE TUNED
HIGH "Q" NOTCHES
FREQUENCY ADJUST
COMBINED
RESPONSES
FREQUENCY ADJUST

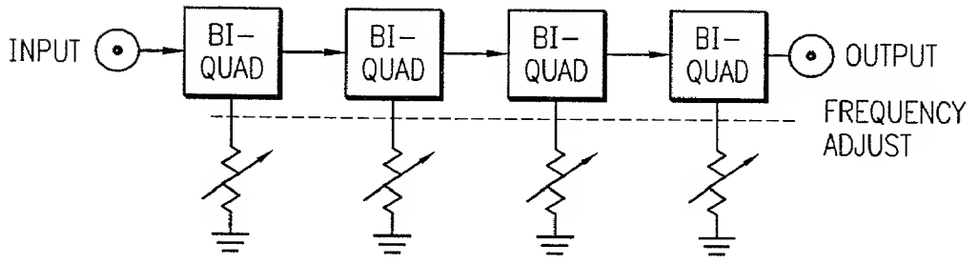

INPUT
BI-
QUAD
BI-
QUAD
BI-
QUAD
BI-
QUAD
OUTPUT
FREQUENCY
ADJUST

FIG. 14

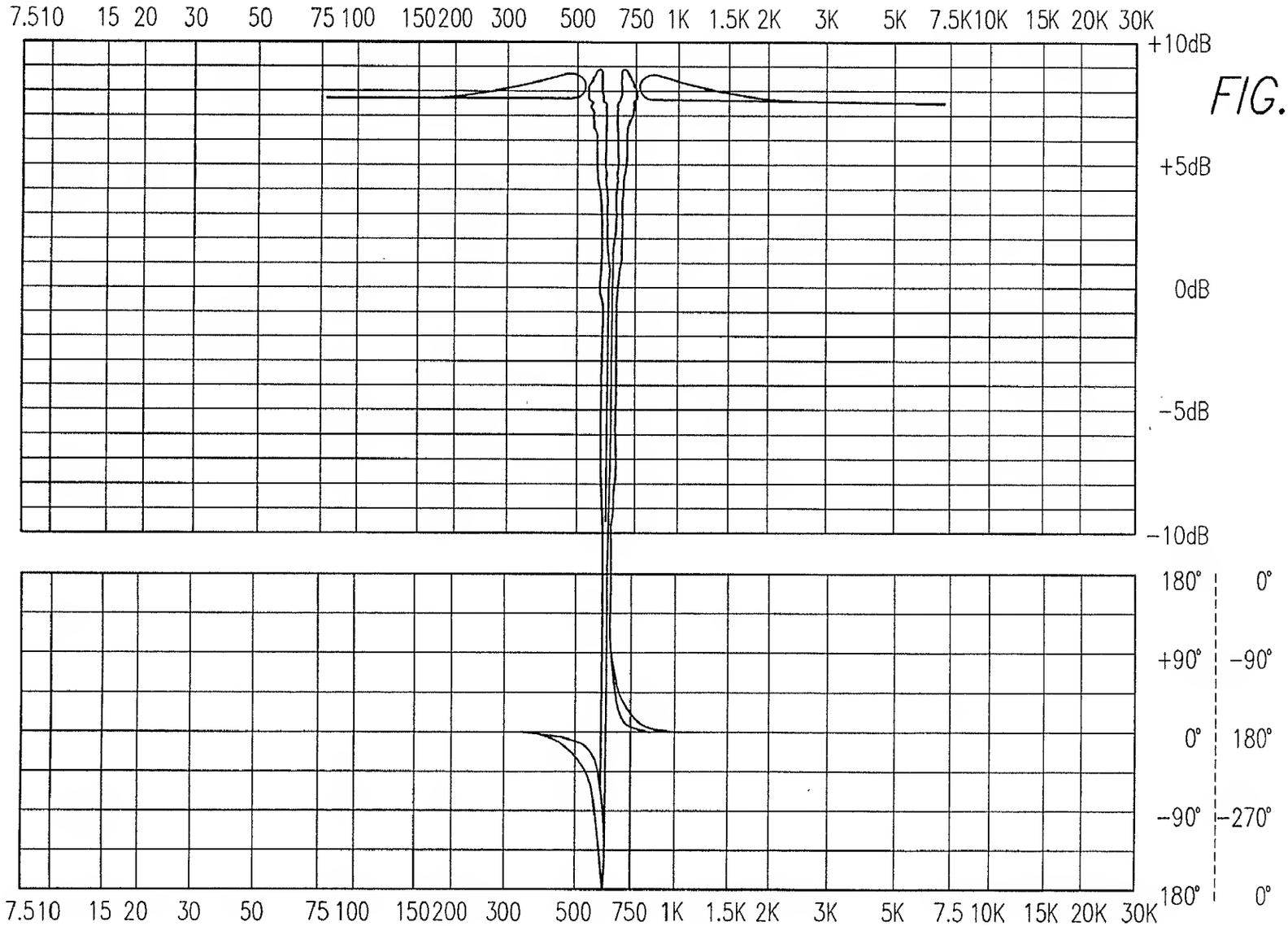
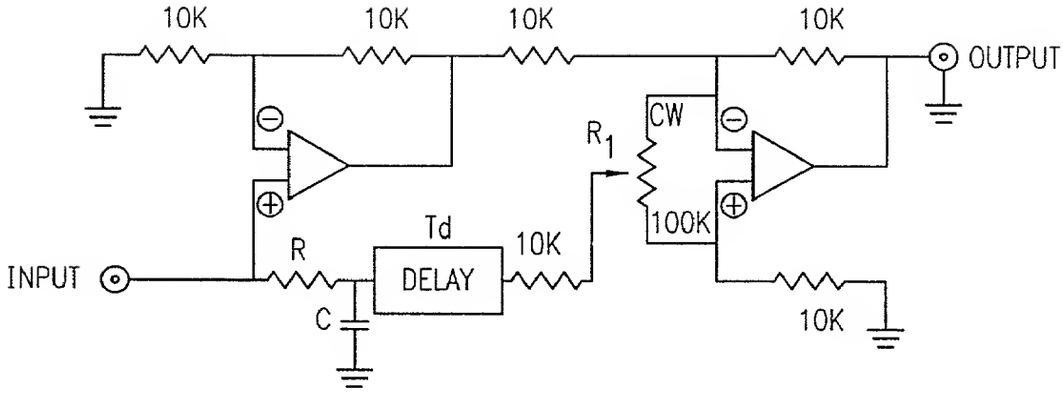


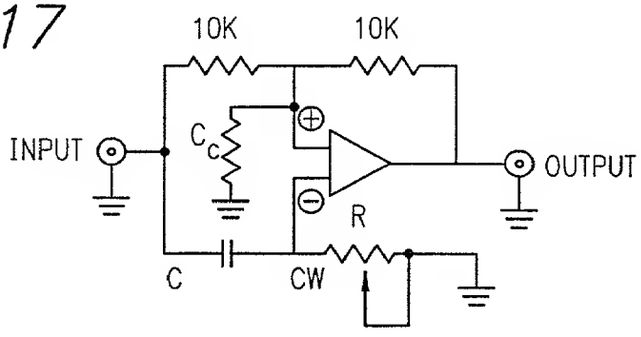
FIG. 15

FIG. 16



R_1 ⊘ CW, CCW

FIG. 17



R ⊘ CW

APPENDIX A

FIGS. A1, A2 and A3 show possible implementations for the low-pass filter (L_P) and the high-pass filter (H_P) cut-off filters. The order of these particular filters is kept small to support a low-MIPS implementation. FIG. A4 shows a possible implementation for the high and low shelving equalizers. These equalizers are used for L_S and H_S. FIG. A5 shows a parametric equalizer implementation usable for boost or cut applications. Combinations of such equalizers can be combined, as shown in FIGS. A6 and A7, to build the compensated notch effect.

Digital Low-Pass Filter Implementation #1

Define sampling frequency

$Fs := 44100$

Define cutoff frequency

$Fc := 14000$

Normalized cutoff frequency

$$fc := 2 \cdot \frac{Fc}{Fs}$$

$fc = 0.635$

Define Filter Q

$q := 10$

$$Q := \frac{1}{q}$$

Define Filter Parameters

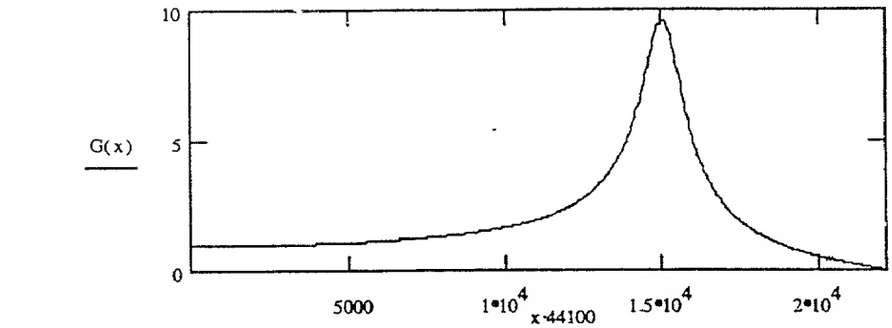
$$F := 2 \cdot \sin\left(fc \cdot \frac{\pi}{2}\right)$$

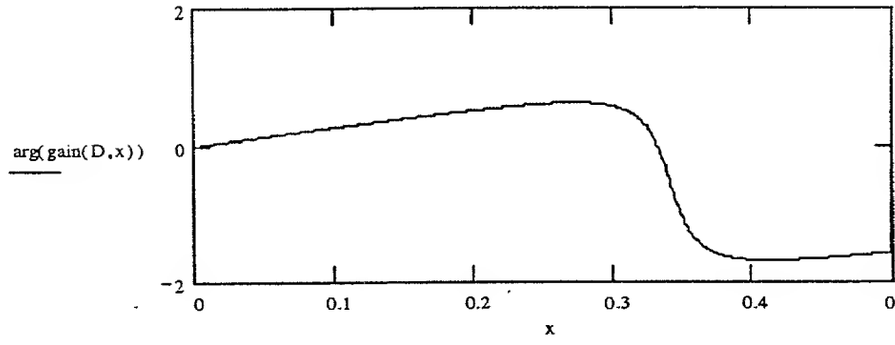
$F = 1.68$

$$A := \frac{F^2}{2}$$

$A = 1.411$

$B := A$





Define Transfer Function (1st column is numerator coefficients in transfer function and 2nd column is denominator coefficients in transfer function). So this filter is a 2nd order IIR.

$$D := \begin{bmatrix} B & 1 \\ B & -(2 - F \cdot Q - F^2) \\ 0 & (1 - F \cdot Q) \end{bmatrix}$$

$$D = \begin{bmatrix} 1.411 & 1 \\ 1.411 & 0.991 \\ 0 & 0.832 \end{bmatrix}$$

Plot frequency and phase response of the filter

$x := 0, .001 .. .5$

$G(x) := |\, gain(D, x) \,|$

0002

FIG. A1

Digital Hi-Pass Filter #1

Define sampling frequency

$Fs := 44100$

Define cutoff frequency

$Fc := 2000$

Normalized cutoff frequency

$$fc := 2 \cdot \frac{Fc}{Fs}$$

Define Filter Q

$q := 2$

$$Q := \frac{1}{q}$$

Define Filter Parameters

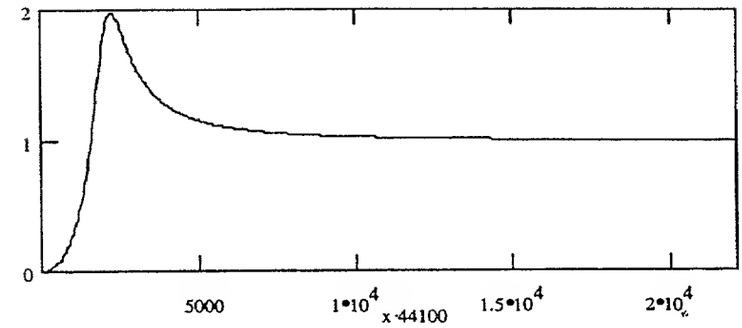
$$F := 2 \cdot \sin\left(fc \cdot \frac{\pi}{2}\right)$$

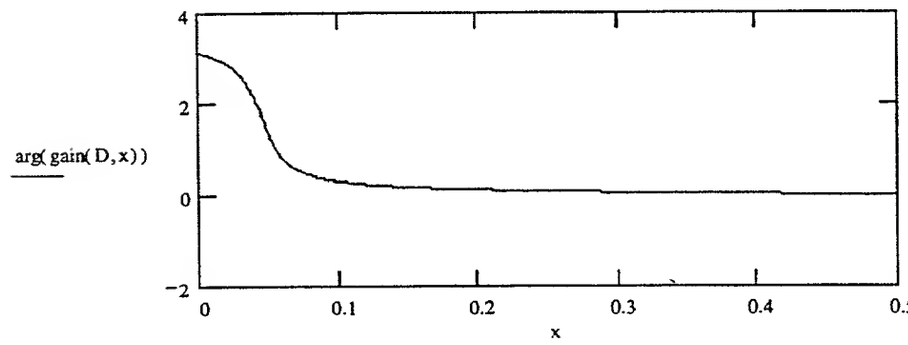
$F = 0.284$

$$A := \frac{F^2}{2}$$

$A = 0.04$

$B := A$





Define Transfer Function (1st column is numerator coefficients in transfer function and 2nd column is denominator coefficients in transfer function). So this filter is a 2nd order IIR.

$$D := \begin{bmatrix} 1 & 1 \\ -2 & -\left(2 - F \cdot Q - F^2\right) \\ 1 & (1 - F \cdot Q) \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & 1 \\ -2 & -1.777 \\ 1 & 0.858 \end{bmatrix}$$

Calculate gain compensation

$$PK := \frac{4}{\left(4 - 2 \cdot F \cdot Q - F^2\right)}$$

$$K := \frac{1}{PK}$$

$K = 0.909$

Plot frequency and phase response of the filter

$x := 0, .001 .. .5$

$G(x) := K \cdot | gain(D, x) |$

0003

FIG. A2

Digital Low-Pass Filter Implementation #2 (one-zero)

Define sampling frequency

$Fs := 44100$

Define cutoff frequency

$Fc := 2000$

Normalized cutoff frequency

$$fc := 2 \cdot \frac{Fc}{Fs}$$

$fc = 0.091$

Define Filter Q

$q := 10$

$$Q := \frac{1}{q}$$

Define Filter Parameters

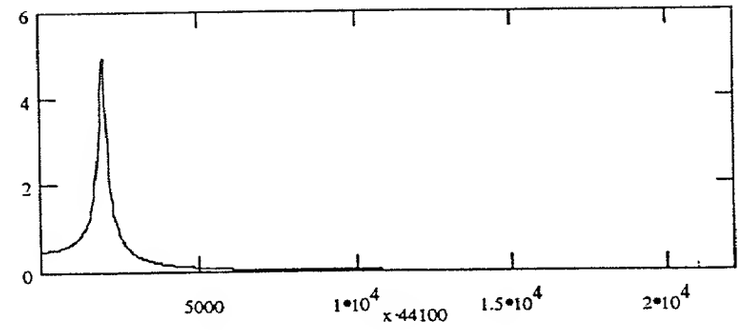
$$F := 2 \cdot \sin\left(fc \cdot \frac{\pi}{2}\right)$$

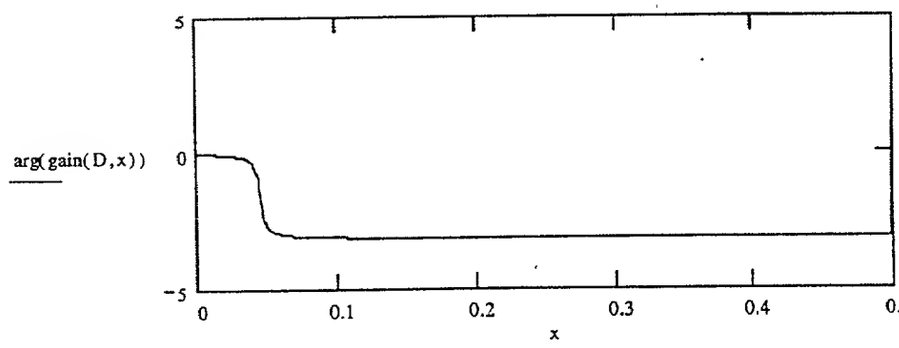
$F = 0.284$

$$A := \frac{F^2}{2}$$

$A = 0.04$

$B := A$





Define Transfer Function (1st column is numerator coefficients in transfer function and 2nd column is denominator coefficients in transfer function). So this filter is a 2nd order IIR.

$$D := \begin{bmatrix} 0 & 1 \\ B & -(2 - F \cdot Q - F^2) \\ 0 & (1 - F \cdot Q) \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 1 \\ 0.04 & -1.891 \\ 0 & 0.972 \end{bmatrix}$$

Plot frequency and phase response of the filter

$x := 0, .001 .. .5$

$G(x) := |\, gain(D, x)\,|$

FIG. A3

Digital Shelving EQ section based on an All-pass filter section

Define sampling frequency

$Fs := 44100$

Define critical frequency

$Fc := 10000$

Normalized critical frequency

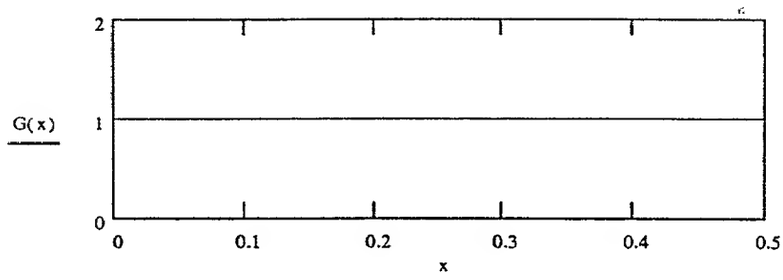
$$wc := 2 \cdot \pi \cdot \frac{Fc}{Fs}$$

Define gain/cut

$k := 4$

Define Filter parameters

$$gamma := \frac{\left(\tan\left(\frac{wc}{2}\right) - 1\right)}{\left(\tan\left(\frac{wc}{2}\right) + 1\right)}$$



Define Transfer Function (1st column is numerator coefficients in transfer function and 2nd column is denominator coefficients in transfer function). So this filter is a 1st order IIR.

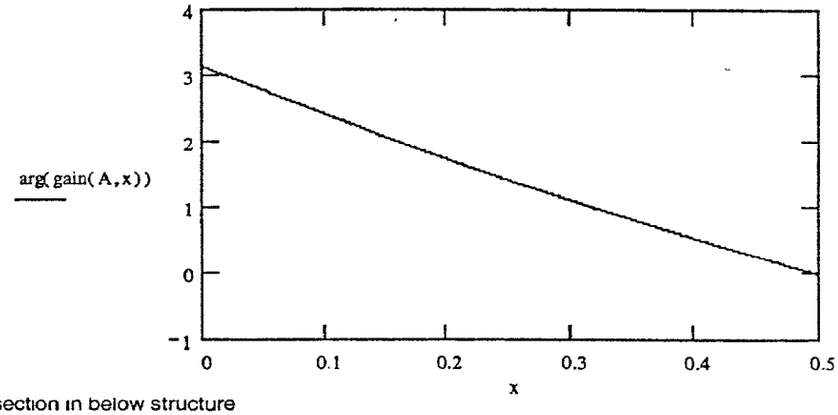
$$A := \begin{bmatrix} -gamma & 1 \\ -1 & gamma \end{bmatrix}$$

$$A = \begin{bmatrix} 0.073 & 1 \\ -1 & -0.073 \end{bmatrix}$$

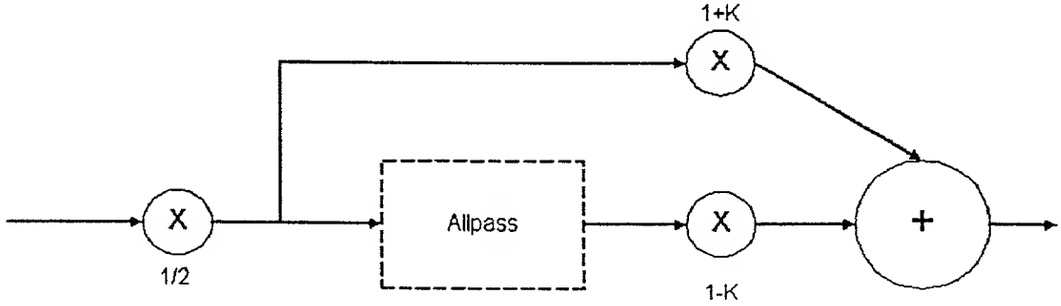
Plot frequency and phase response of the filter

$x := 0, .001 .. 0.5$

$G(x) := |\, gain(A, x) \,|$



To produce desired parametric EQ embed all-pass section in below structure



In a DSP implementation, the filter output is computed as per the above signal flow. Below we calculate the overall transfer function of the above system by transforming the numerator of the all-pass section as necessary.

FIG. A4 (Sheet 1 of 2)

$D := A$

$G := D^{\langle 1 \rangle}$ extract denominator

$F := D^{\langle 0 \rangle}$ extract numerator

$$M := \frac{(1-k)}{2}$$

$$L := \frac{(1+k)}{2}$$

build new numerator:

$F_0 := M \cdot F_0 + L$

$F_1 := M \cdot F_1 + L \cdot G_1$

$D^{\langle 0 \rangle} := F$

$$D = \begin{bmatrix} 2.39 & 1 \\ 1.317 & -0.073 \end{bmatrix}$$

Plot magnitude response:

$|\underline{\text{gain}}(D,x)|$

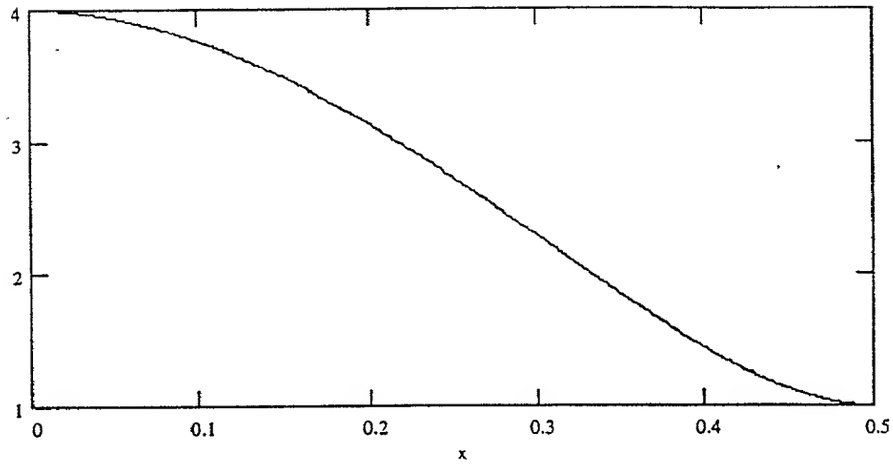


FIG. A4 (Sheet 2 of 2)

Digital Notch/Parametric EQ section based on an all-pass section

Define sampling frequency

$Fs := 44100$

Define critical frequency

$Fc := 5500$

Normalized critical frequency

$$wc := 2\cdot\pi\cdot\frac{Fc}{Fs}$$

Define Filter Q

$Q := 4$

Define gain/cut

$k := 4$

Define Filter parameters

$gamma := -\cos(wc)$

$$fbeta := \frac{wc}{2\cdot Q}$$

$$fbeta := if\left(fbeta > \frac{\pi}{4}, \frac{\pi}{4}, fbeta\right)$$

$$beta := \frac{1-\tan(fbeta)}{1+\tan(fbeta)}$$

Define Transfer Function (1st column is numerator c and 2nd column is denominator coefficients in transf order IIR.

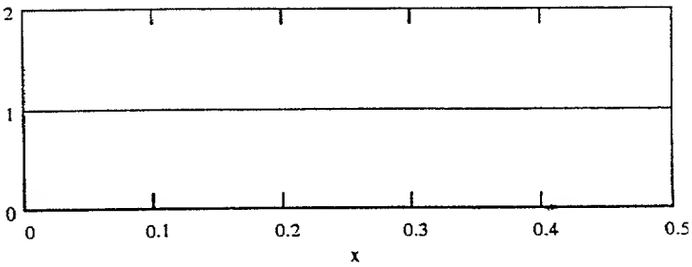
$$A := \begin{bmatrix} beta & 1 \\ gamma\cdot(1+beta) & gamma\cdot(1+beta) \\ 1 & beta \end{bmatrix}$$

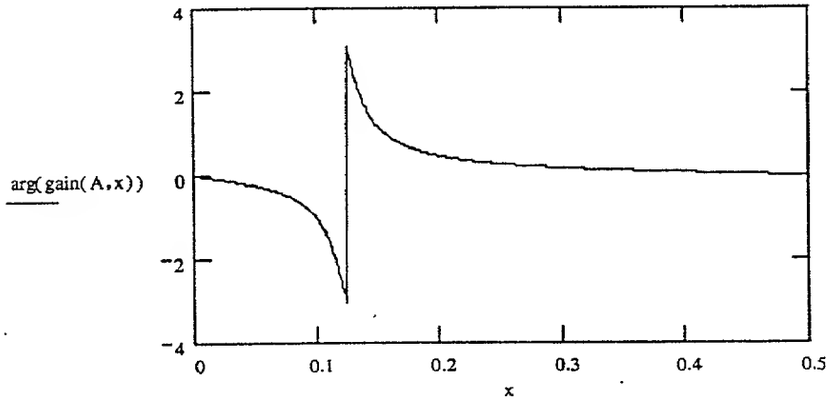
$$A = \begin{bmatrix} 0.821 & 1 \\ -1.29 & -1.29 \\ 1 & 0.821 \end{bmatrix}$$

Plot frequency and phase response of the filter

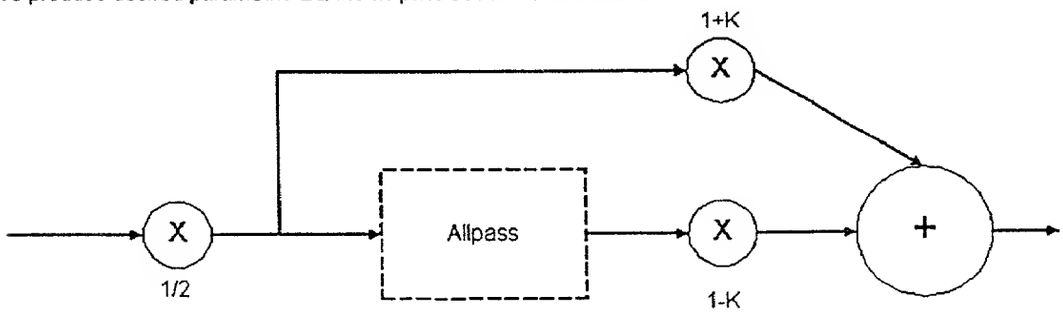
$x := 0, .001 .. 0.5$

$G(x) := |gain(A, x)|$





To produce desired parametric EQ the all-pass section is embedded in the below structure:



In a DSP implementation, the filter output is computed as per the above signal flow. Below we calculate the overall transfer function of the above system by

FIG. A5 (Sheet 1 of 2)

0007

transforming the numerator of the all-pass section as necessary.

$D := A$

$G := D^{\langle 1 \rangle}$ extract denominator

$F := D^{\langle 0 \rangle}$ extract numerator

$$M := \frac{(1-k)}{2}$$

$$L := \frac{(1+k)}{2}$$

build new numerator:

$F_0 := M \cdot F_0 + L$

$F_1 := M \cdot F_1 + L \cdot G_1$

$F_2 := M \cdot F_2 + L \cdot G_2$

$D^{\langle 0 \rangle} := F$

$$D = \begin{bmatrix} 1.268 & 1 \\ -1.29 & -1.29 \\ 0.553 & 0.821 \end{bmatrix}$$

Magnitude Response:

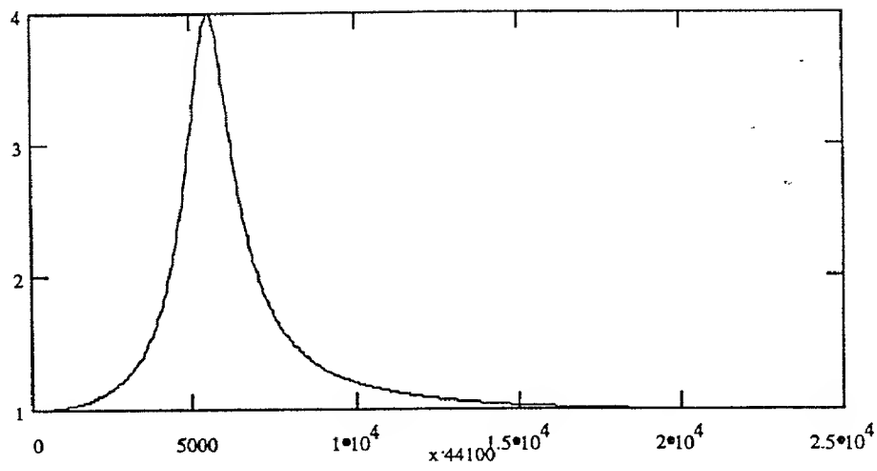


FIG. A5 (Sheet 2 of 2)

Digital Implementation of a Weighted Notch #1

Consists of 3 parametric EQ section designed using the procedure described in
"Notch/Parametric EQ section based on an all-pass section"

2 low Q gain section surround the critical frequency notch.

For compensation section #1 and #2 (D1 and D2 respectively) Q is 2 and k is 1.5
resulting in the below transfer functions. The resulting filters are two 2nd order IIR filters where column 0 represents the numerator and column 1 represents the denominator.

$$D1 := \begin{bmatrix} 1.076 & 1 \\ -1.283 & -1.283 \\ 0.619 & 0.695 \end{bmatrix}$$

$$D2 := \begin{bmatrix} 1.089 & 1 \\ -1.079 & -1.079 \\ 0.554 & 0.643 \end{bmatrix}$$

For critical frequency section #3 (D3 below) Q is 10 and k is 0. The resulting filter is a 2nd order IIR filters where column 0 represents the numerator and column 1 represents the denominator.

$$D3 := \begin{bmatrix} 0.962 & 1 \\ -1.363 & -1.363 \\ 0.962 & 0.925 \end{bmatrix}$$

Plot of the overall frequency response of the 3 IIR filters cascaded

$x := 0, .001 .. 0.5$

$G1(x) := |\, gain(D1, x) \,|$

$G2(x) := |\, gain(D2, x) \,|$

$G3(x) := |\, gain(D3, x) \,|$

$H(x) := 20 \cdot \log(G1(x) \cdot G2(x) \cdot G3(x))$

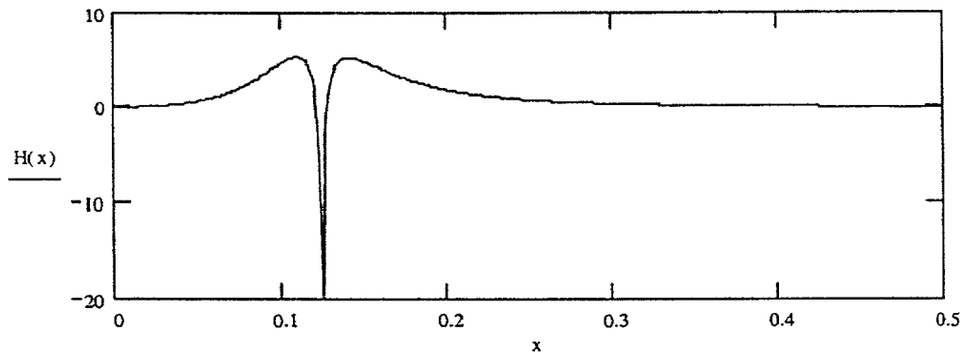


FIG. A6

Digital Implementation of a Weighted Notch #2

Consists of 3 parametric EQ section designed using the procedure described in
"Notch/Parametric EQ section based on an all-pass section"

One medium Q gain compensation section centered around two notches.

For compensation section #1 (D1 below) Q is 4 and k is 4 resulting in the below transfer function. The resulting filters is a 2nd order IIR filters where column 0 represents the numerator and column 1 represents the denominator.

$$D1 := \begin{bmatrix} 1.268 & 1 \\ -1.29 & -1.29 \\ 0.553 & 0.821 \end{bmatrix}$$

For critical frequency notches sections #2 and #3 (D2 and D3 below respectively) Q is 10 and k is 0. The resulting filters are two 2nd order IIR filters where column 0 represents the numerator and column 1 represents the denominator.

$$D2 := \begin{bmatrix} 0.963 & 1 \\ -1.383 & -1.383 \\ 0.963 & 0.926 \end{bmatrix}$$

$$D3 := \begin{bmatrix} 0.962 & 1 \\ -1.343 & -1.343 \\ 0.962 & 0.923 \end{bmatrix}$$

Plot of the overall frequency response of the 3 IIR filters cascaded:

$x := 0, .0001 .. 0.5$

$G1(x) := |\, gain(D1, x) \,|$

$G2(x) := |\, gain(D2, x) \,|$

$G3(x) := |\, gain(D3, x) \,|$

$H(x) := 20 \cdot \log(G1(x) \cdot G2(x) \cdot G3(x))$ (cascaded response)

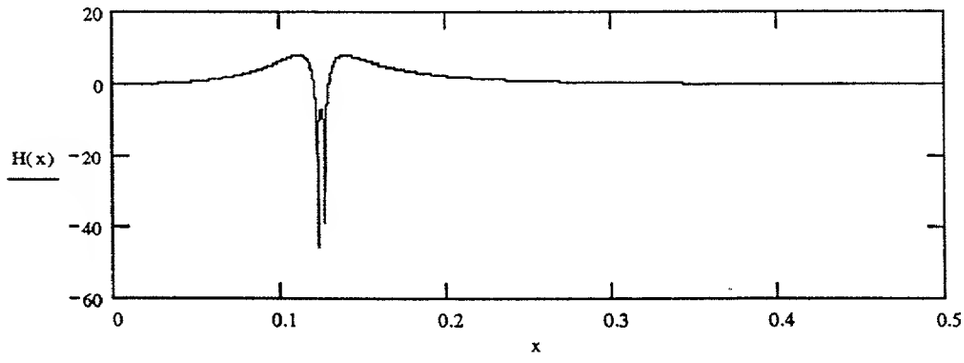


FIG. A7

APPENDIX B

Figure B1 is a block diagram of a compensation system used to compensate a small bookshelf speaker having a 5-inch bass driver and 3-inch tweeter. The speaker is characterized as follows:

Woofer: dome = 4.33 k (notch with LC)
cone = 4.9 k (compensated notch only)
edge of cone = 2.8 k (active notch)

Tweeter: dome = 15.1 k (no compensation)
cone cr = 11.6 k (active notch)
cone cr = 9.32 k / 9.45 k (damp compound)
W_0 = 1.38 k (LCR notch)

Fourteen adjustments are provided. FIG. B2 shows circuits used to create L_X, H_X, L_S, H_S, L_P, and H_P filters and parameters adjustments. A lower frequency cutoff peak was added to create response character of an 8 inch bookshelf system. For this circuit, Q and peaking amplitude are set by the components marked by square boxes. FIGS. B3 and B4 show W_0 notch sections and representative tuning for the speaker. W_0... and |A| in dB are adjustable, while a resistor set Q. Two groups of these make four adjustments. FIG. B5 shows a single low-Q boost. Adjustments for boost and frequency are provided by the op amp section. FIG. B6 is an all-pass equalizer used for time correction. The all-pass equalizer combines outputs from the six active process circuits.

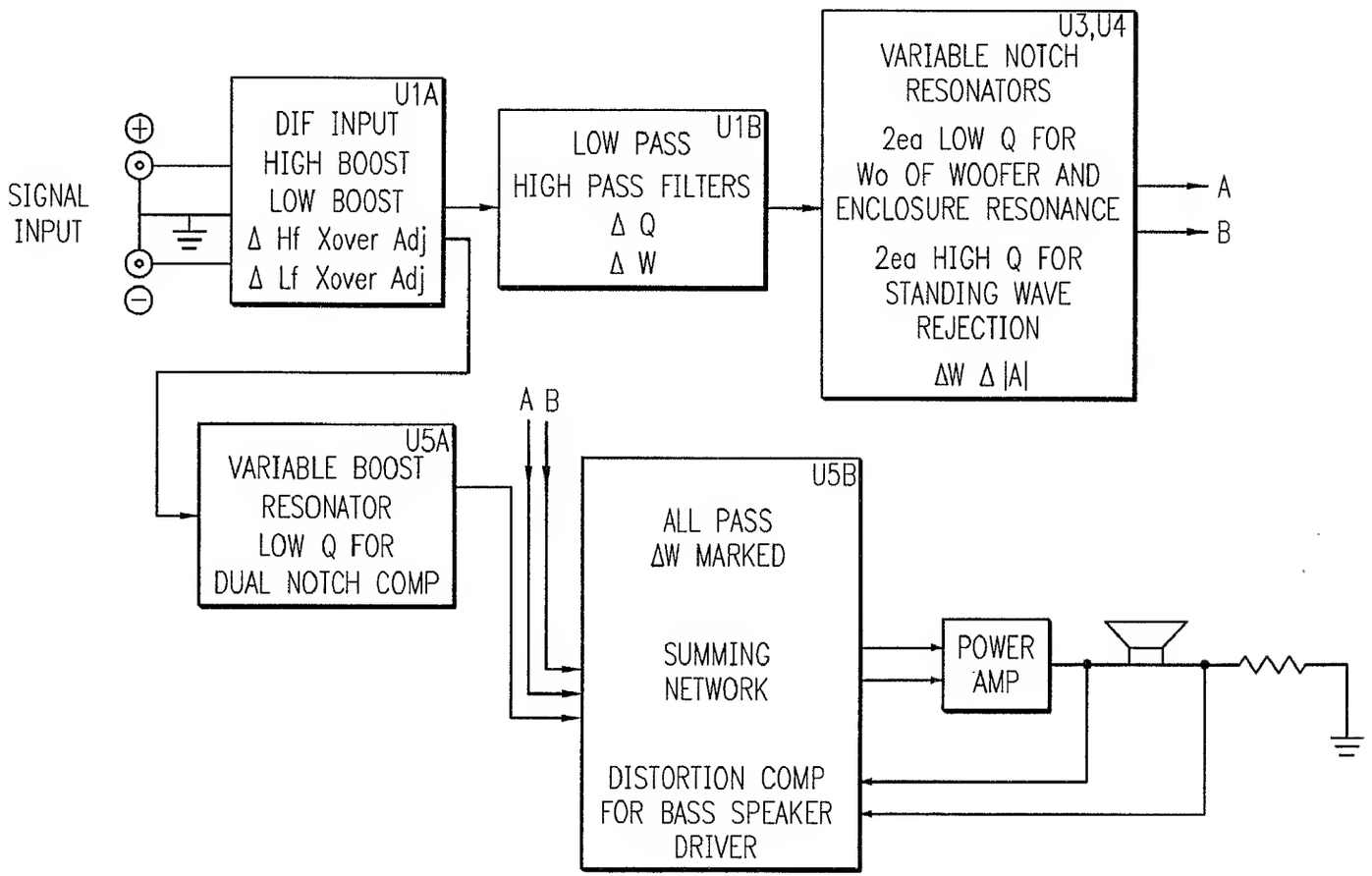

SIGNAL INPUT
U1A
DIF INPUT
HIGH BOOST
LOW BOOST
Δ Hf Xover Adj
Δ Lf Xover Adj
U1B
LOW PASS
HIGH PASS FILTERS
Δ Q
Δ W
U3,U4
VARIABLE NOTCH RESONATORS
2ea LOW Q FOR Wo OF WOOFER AND ENCLOSURE RESONANCE
2ea HIGH Q FOR STANDING WAVE REJECTION
ΔW Δ |A|
A
B
U5A
VARIABLE BOOST RESONATOR
LOW Q FOR DUAL NOTCH COMP
A B
U5B
ALL PASS
ΔW MARKED
SUMMING NETWORK
DISTORTION COMP FOR BASS SPEAKER DRIVER
POWER AMP

FIG. B1

2000

SLOPE EQ AND BANDPASS FILTER ADJUSTED
W_L=65Hz, 44Hz (PARALLEL CAPS)
W_H=17kHz
W_{LX}=400Hz -> 20Hz
W_{HX}=3kHz -> 25kHz

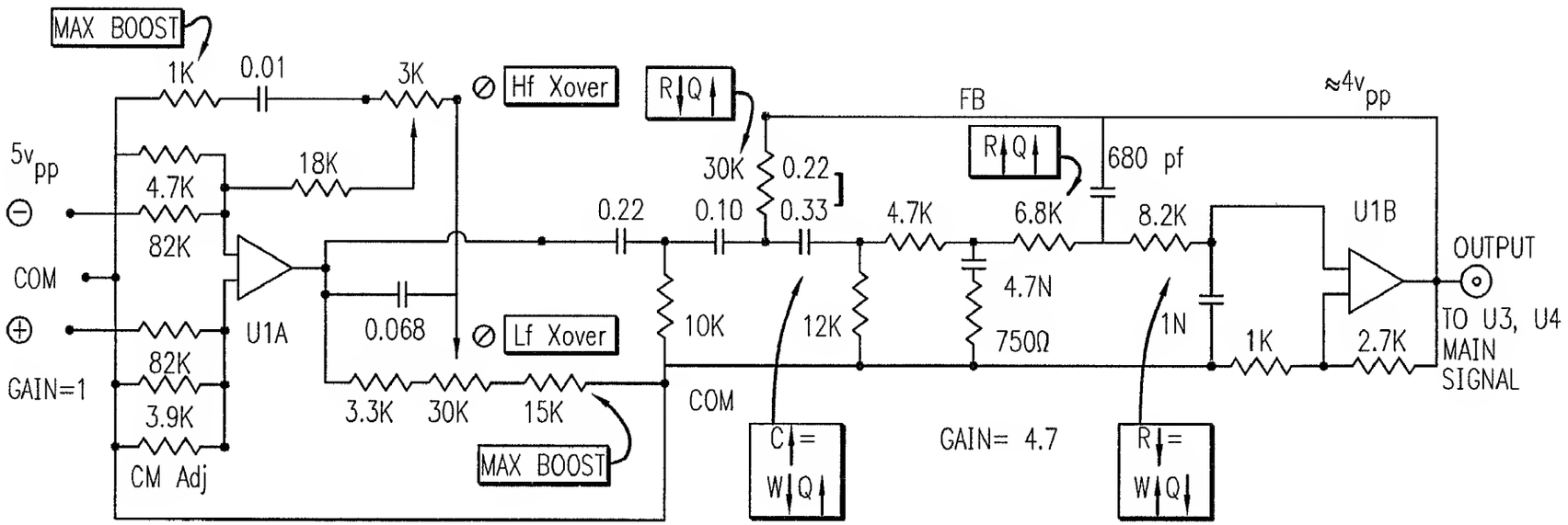
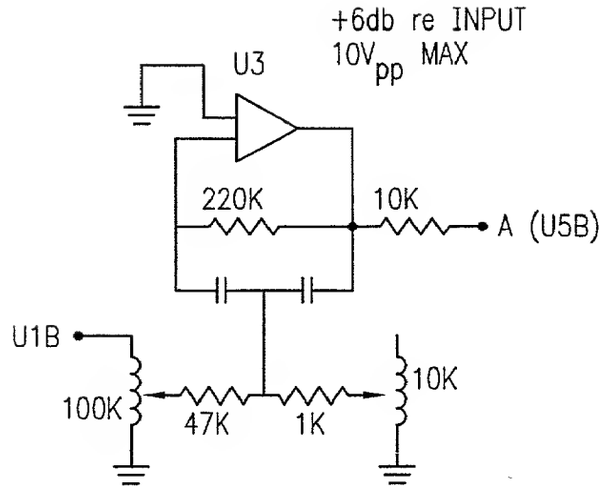


FIG. B2

FIG. B3



0.0015=3.2-3.6k (2.5k->6.8k)
0.0033=1.4->1.7k (1.2k->3.3k)
0.0047≅900-1.2k (800->2.2k)
0.15=120 (80-220)

LOWQ R|A|=47k||33k
RQ=68k||100k

V_{OUT}=10V_{pp} MAX
(+6dB)

FIG. B4

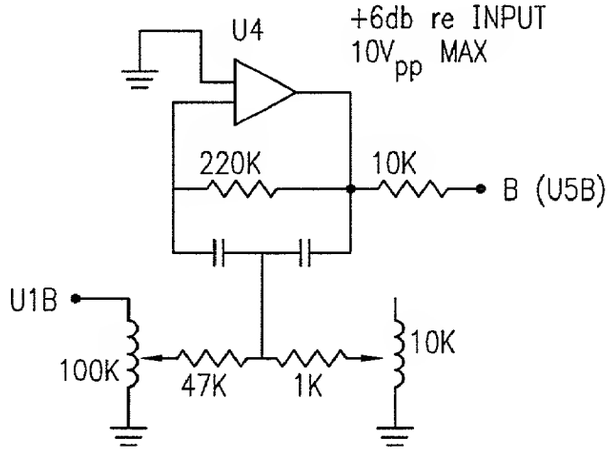
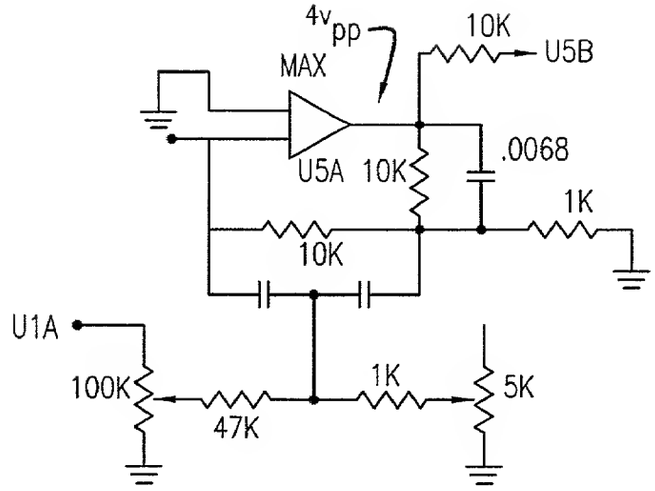


FIG. B5



0.033=270->1.4k
0.0068

700Hz opt
@ 3/4 max W_0

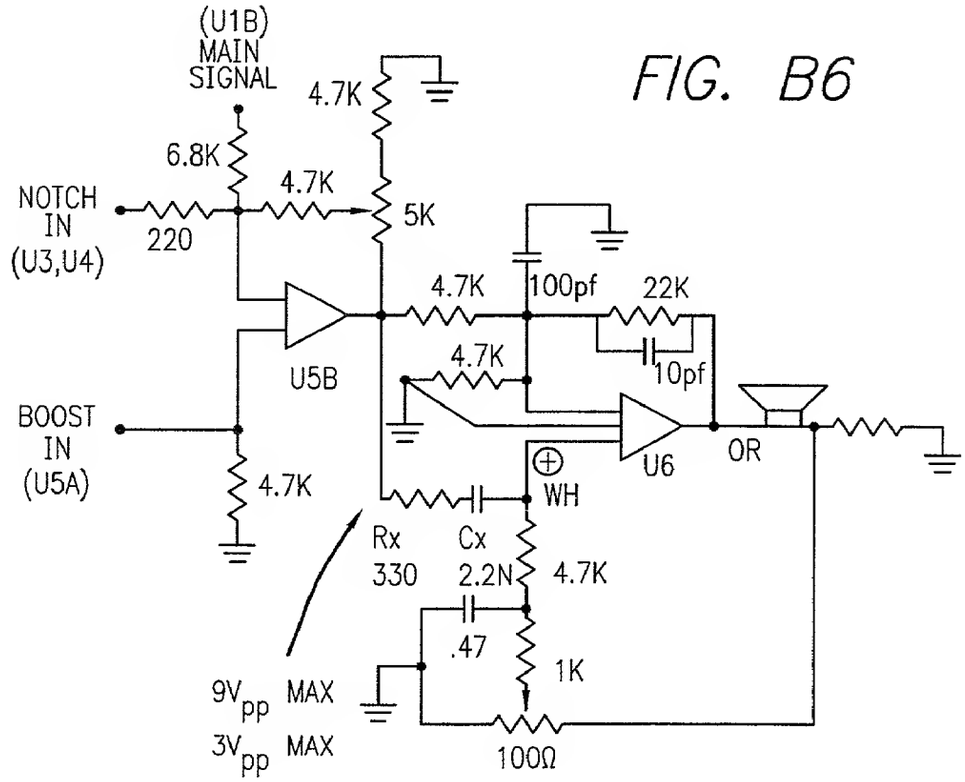


FIG. B6

APPENDIX C

A computer adjusted DSP compensator was implemented using a standard PC and a Motorola EVM56362 DSP evaluation board. Source code and application notes follow.

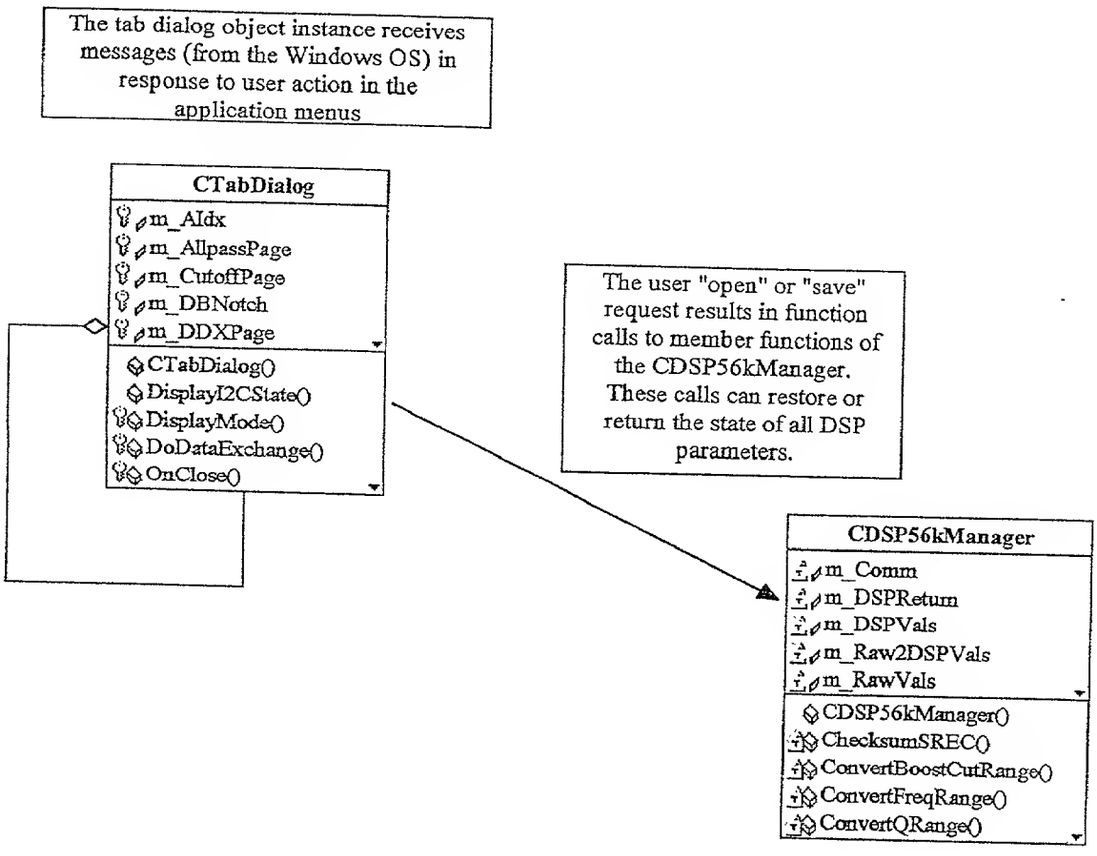
Speaker Tuning Application Note

The speaker tuning application is written as a Windows OS 32-bit application using the object-oriented MFC application framework (please refer to the documentation of MFC that is included with Microsoft Visual C++ development environment). The application has a dialog interface. The main dialog class, CTabDialog, is a subclass of the MFC CDialog class. The CTabDialog class implements a "tabbed" dialog interface. Each tab in the dialog is a subclass of the MFC CPropertyPage class. Each tab represents different aspects of the speaker correction algorithm. The following is a list of tab classes:

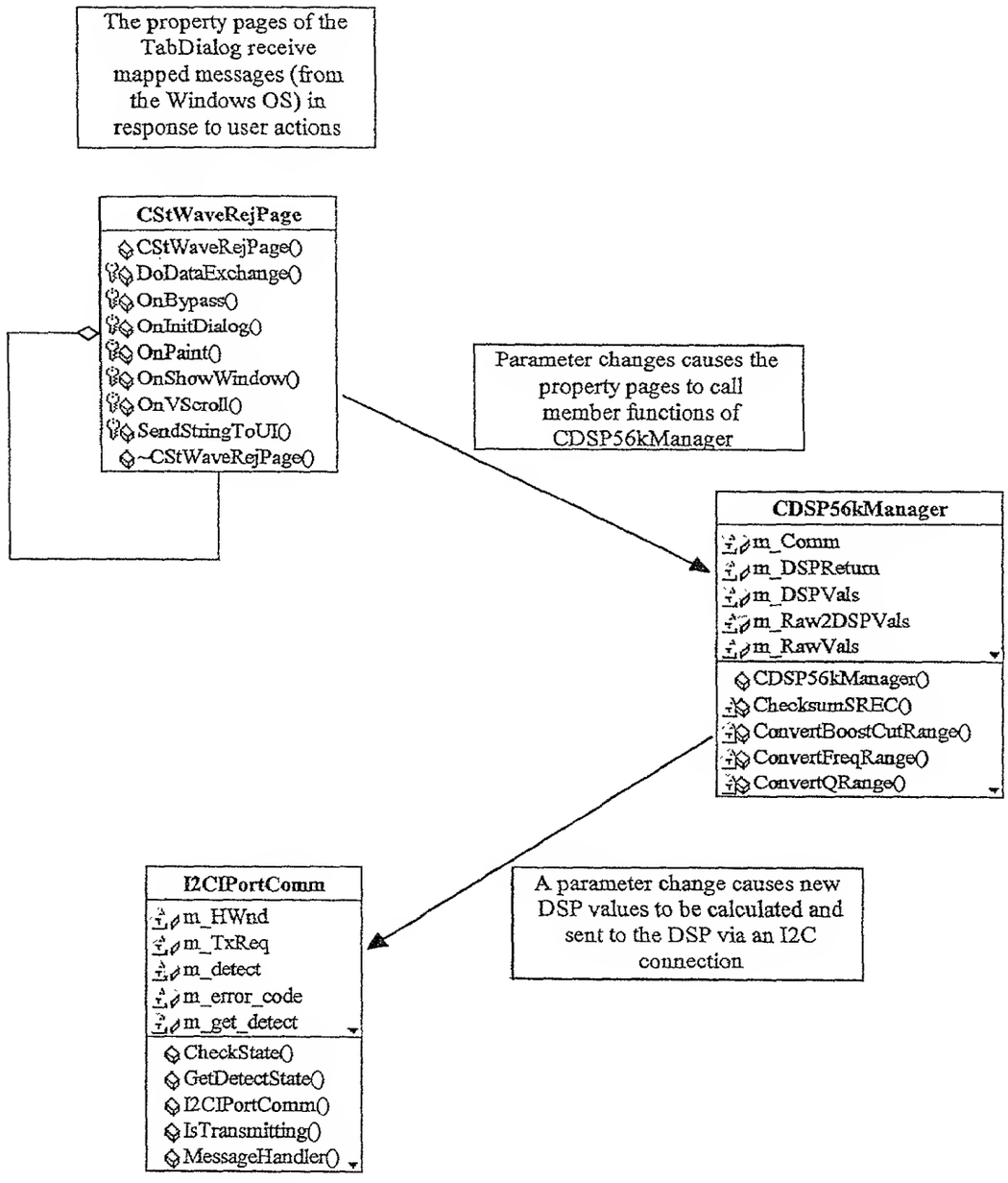
- CMainPage which implements UI for pre and post volume controls amongst other things
- CShelvPage which implements UI controls for low and high shelving equalization filters
- CCutoffPage which implements UI controls for low and high peaking cutoff filters
- CNotchPage1 and CNotchPage2 which implements UI controls for a number of notch filters (to for example limit resonance in the speaker)
- CStWaveRejectPage which implements UI controls for a set of filters which can limit standing waves in the speaker cabinet
- CDBNotch which implements UI controls for a double-tuned notch filter
- CAllpassPage which implements UI for a 2^{nd}-order allpass filter

Each adjustable parameter on a tab page is represented by an instance of the CSlider class. Each UI slider has a range of 4096 discrete values. When a user manipulates a UI slider a message is sent from the tab page to an instance of the CDSP56Manager class. In this class, the appropriate calculation takes place to transform the linear input value into one or more values necessary to compute the transfer function represented by the user settings. These computed values are transmitted to the DSP using an I2C serial connection. The DSP executes the calculations necessary for the real-time implementation of the above-mentioned transfer function. The DSP is capable of computing a series of filter calculations in real-time to allow the total cascaded transfer function of all the speaker correction filters to be realized. The DSP can receive analog or digital input data and transmit processed analog or digital output data.

Speaker Parameter File Input and Output Logic



Example of a Property Page UI to DSP connection logic



```
// COMPortChooser.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "COMPortChooser.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCOMPortChooser dialog


CCOMPortChooser::CCOMPortChooser(CWnd* pParent /*=NULL*/)
  : CDialog(CCOMPortChooser::IDD, pParent)
{
  //{{AFX_DATA_INIT(CCOMPortChooser)
    // NOTE: the ClassWizard will add member initialization here
  //}}AFX_DATA_INIT
}


void CCOMPortChooser::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CCOMPortChooser)
    // NOTE: the ClassWizard will add DDX and DDV calls here
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CCOMPortChooser, CDialog)
  //{{AFX_MSG_MAP(CCOMPortChooser)
  ON_BN_CLICKED(IDC_RADIO1, OnRadio1)
  ON_BN_CLICKED(IDC_RADIO2, OnRadio2)
  ON_BN_CLICKED(IDC_RADIO3, OnRadio3)
  ON_BN_CLICKED(IDC_RADIO4, OnRadio4)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCOMPortChooser message handlers

void CCOMPortChooser::OnRadio1()
{
  // TODO: Add your control notification handler code here
  which_port = 1;
}

void CCOMPortChooser::OnRadio2()
{
  // TODO: Add your control notification handler code here
  which_port = 2;

}

void CCOMPortChooser::OnRadio3()
{
  // TODO: Add your control notification handler code here
  which_port = 3;

}

void CCOMPortChooser::OnRadio4()
{
  // TODO: Add your control notification handler code here
  which_port = 4;
```

0005

```
}

BOOL CCOMPortChooser::OnInitDialog()
{
  CDialog::OnInitDialog();

  // TODO: Add extra initialization here
  which_port = 3;

  return TRUE;  // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
// CutoffPage.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "CutoffPage.h"
#include "DSP56kManager.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCutoffPage property page

IMPLEMENT_DYNCREATE(CCutoffPage, CPropertyPage)

CCutoffPage::CCutoffPage() : CPropertyPage(CCutoffPage::IDD)
{
  //{{AFX_DATA_INIT(CCutoffPage)
  //}}AFX_DATA_INIT
}

CCutoffPage::~CCutoffPage()
{
}

void CCutoffPage::DoDataExchange(CDataExchange* pDX)
{
  CPropertyPage::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CCutoffPage)
  DDX_Control(pDX, IDC_CHECK6, m_BypassSecondButton);
  DDX_Control(pDX, IDC_CHECK5, m_BypassFirstButton);
  DDX_Control(pDX, IDC_SLIDER6, m_HiBoostSlider);
  DDX_Control(pDX, IDC_SLIDER5, m_HiQSlider);
  DDX_Control(pDX, IDC_SLIDER4, m_HiFreqSlider);
  DDX_Control(pDX, IDC_SLIDER3, m_LoBoostSlider);
  DDX_Control(pDX, IDC_SLIDER2, m_LoQSlider);
  DDX_Control(pDX, IDC_SLIDER1, m_LoFreqSlider);
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CCutoffPage, CPropertyPage)
  //{{AFX_MSG_MAP(CCutoffPage)
  ON_WM_VSCROLL()
  ON_BN_CLICKED(IDC_CHECK5, OnBypassFirst)
  ON_BN_CLICKED(IDC_CHECK6, OnBypassSecond)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCutoffPage message handlers

BOOL CCutoffPage::OnInitDialog()
{
  CPropertyPage::OnInitDialog();

  // TODO: Add extra initialization here
  m_LoFreqSlider.SetRange(0,CONTROL_RANGE);
  m_LoFreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch6Freq));
  m_LoFreqSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_LoQSlider.SetRange(0,CONTROL_RANGE);
  m_LoQSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch6Q));
  m_LoQSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_LoBoostSlider.SetRange(0,CONTROL_RANGE);
  m_LoBoostSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch6Boost));
  m_LoBoostSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_HiFreqSlider.SetRange(0,CONTROL_RANGE);
  m_HiFreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch7Freq));
```

0007

```
  m_HiFreqSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_HiQSlider.SetRange(0,CONTROL_RANGE);
  m_HiQSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch7Q));
  m_HiQSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_HiBoostSlider.SetRange(0,CONTROL_RANGE);
  m_HiBoostSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch7Boost));
  m_HiBoostSlider.SetTicFreq((CONTROL_RANGE+1)/16);

//  g_DSPManager->SetCutoff1Freq(CONTROL_RANGE-m_LoFreqSlider.GetPos());
//  g_DSPManager->SetCutoff1Q(CONTROL_RANGE-m_LoQSlider.GetPos());
//  g_DSPManager->SetCutoff1Boost(CONTROL_RANGE-m_LoBoostSlider.GetPos());
//  g_DSPManager->SetCutoff2Freq(CONTROL_RANGE-m_HiFreqSlider.GetPos());
//  g_DSPManager->SetCutoff2Q(CONTROL_RANGE-m_HiQSlider.GetPos());
//  g_DSPManager->SetCutoff2Boost(CONTROL_RANGE-m_HiBoostSlider.GetPos());


  return TRUE;  // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CCutoffPage::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
  // TODO: Add your message handler code here and/or call default

  CPropertyPage::OnVScroll(nSBCode, nPos, pScrollBar);

  Sleep(50);
  if( (CSliderCtrl *)pScrollBar == &m_LoFreqSlider )
    g_DSPManager->SetCutoff1Freq(CONTROL_RANGE-m_LoFreqSlider.GetPos());
  else if( (CSliderCtrl *)pScrollBar == &m_LoQSlider )
    g_DSPManager->SetCutoff1Q(CONTROL_RANGE-m_LoQSlider.GetPos());
  else if( (CSliderCtrl *)pScrollBar == &m_LoBoostSlider )
    g_DSPManager->SetCutoff1Boost(CONTROL_RANGE-m_LoBoostSlider.GetPos());
  else if( (CSliderCtrl *)pScrollBar == &m_HiFreqSlider )
    g_DSPManager->SetCutoff2Freq(CONTROL_RANGE-m_HiFreqSlider.GetPos());
  else if( (CSliderCtrl *)pScrollBar == &m_HiQSlider )
    g_DSPManager->SetCutoff2Q(CONTROL_RANGE-m_HiQSlider.GetPos());
  else if( (CSliderCtrl *)pScrollBar == &m_HiBoostSlider )
    g_DSPManager->SetCutoff2Boost(CONTROL_RANGE-m_HiBoostSlider.GetPos());

}

void CCutoffPage::OnBypassFirst()
{
  // TODO: Add your control notification handler code here
  int state = m_BypassFirstButton.GetState() & 0x3;

  g_DSPManager->SetBypassSection(state,kBypassHipass);

}

void CCutoffPage::OnBypassSecond()
{
  // TODO: Add your control notification handler code here
  int state = m_BypassSecondButton.GetState() & 0x3;

  g_DSPManager->SetBypassSection(state,kBypassLopass);

}
```

```
#if !defined(AFX_COMPORTCHOOSER_H__B5D510E7_F7D5_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_COMPORTCHOOSER_H__B5D510E7_F7D5_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// COMPortChooser.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CCOMPortChooser dialog

class CCOMPortChooser : public CDialog
{

// Construction
public:
  CCOMPortChooser(CWnd* pParent = NULL);   // standard constructor

  int which_port;

// Dialog Data
  //{{AFX_DATA(CCOMPortChooser)
  enum { IDD = IDD_DIALOG3 };
    // NOTE: the ClassWizard will add data members here
  //}}AFX_DATA


// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CCOMPortChooser)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:

  // Generated message map functions
  //{{AFX_MSG(CCOMPortChooser)
  afx_msg void OnRadio1();
  afx_msg void OnRadio2();
  afx_msg void OnRadio3();
  afx_msg void OnRadio4();
  virtual BOOL OnInitDialog();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_COMPORTCHOOSER_H__B5D510E7_F7D5_11D2_96EE_006097CDB9E2__INCLUDED_)
```

```
#if !defined(AFX_CUTOFFPAGE_H__6F151625_D84D_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_CUTOFFPAGE_H__6F151625_D84D_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// CutoffPage.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CCutoffPage dialog

class CCutoffPage : public CPropertyPage
{
  DECLARE_DYNCREATE(CCutoffPage)

// Construction
public:
  CCutoffPage();
  ~CCutoffPage();

// Dialog Data
  //{{AFX_DATA(CCutoffPage)
  enum { IDD = IDD_PP7 };
  CButton m_BypassSecondButton;
  CButton m_BypassFirstButton;
  CSliderCtrl m_HiBoostSlider;
  CSliderCtrl m_HiQSlider;
  CSliderCtrl m_HiFreqSlider;
  CSliderCtrl m_LoBoostSlider;
  CSliderCtrl m_LoQSlider;
  CSliderCtrl m_LoFreqSlider;
  //}}AFX_DATA


// Overrides
  // ClassWizard generate virtual function overrides
  //{{AFX_VIRTUAL(CCutoffPage)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  // Generated message map functions
  //{{AFX_MSG(CCutoffPage)
  virtual BOOL OnInitDialog();
  afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
  afx_msg void OnBypassFirst();
  afx_msg void OnBypassSecond();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_CUTOFFPAGE_H__6F151625_D84D_11D2_96EE_006097CDB9E2__INCLUDED_)
```

0010

```
#if !defined(AFX_ALLPASSPAGE_H__F4767B24_11B3_11D3_96EE_006097CDB9E2__INCLUDED_)
#define AFX_ALLPASSPAGE_H__F4767B24_11B3_11D3_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// AllpassPage.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CAllpassPage dialog

class CTabDialog;

class CAllpassPage : public CPropertyPage
{
  DECLARE_DYNCREATE(CAllpassPage)

// Construction
public:
  CAllpassPage();
  ~CAllpassPage();

// Dialog Data
  //{{AFX_DATA(CAllpassPage)
  enum { IDD = IDD_PP10 };
  CButton m_BypassButton;
  CSliderCtrl m_QSlider;
  CSliderCtrl m_FrequencySlider;
  //}}AFX_DATA


// Overrides
  // ClassWizard generate virtual function overrides
  //{{AFX_VIRTUAL(CAllpassPage)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  // Generated message map functions
  //{{AFX_MSG(CAllpassPage)
  afx_msg void OnBypass();
  virtual BOOL OnInitDialog();
  afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
  afx_msg void OnPaint();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()

  CTabDialog *m_ParentWindow;
  void SendStringToUI(int which);

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_ALLPASSPAGE_H__F4767B24_11B3_11D3_96EE_006097CDB9E2__INCLUDED_)
```

0011

```
// AllpassPage.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "AllpassPage.h"
#include "TabDialog.h"
#include "DSP56kManager.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAllpassPage property page

IMPLEMENT_DYNCREATE(CAllpassPage, CPropertyPage)

CAllpassPage::CAllpassPage() : CPropertyPage(CAllpassPage::IDD)
{
  //{{AFX_DATA_INIT(CAllpassPage)
  //}}AFX_DATA_INIT
}

CAllpassPage::~CAllpassPage()
{
}

void CAllpassPage::DoDataExchange(CDataExchange* pDX)
{
  CPropertyPage::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CAllpassPage)
  DDX_Control(pDX, IDC_CHECK1, m_BypassButton);
  DDX_Control(pDX, IDC_SLIDER3, m_QSlider);
  DDX_Control(pDX, IDC_SLIDER1, m_FrequencySlider);
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CAllpassPage, CPropertyPage)
  //{{AFX_MSG_MAP(CAllpassPage)
  ON_BN_CLICKED(IDC_CHECK1, OnBypass)
  ON_WM_VSCROLL()
  ON_WM_PAINT()
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAllpassPage message handlers

void CAllpassPage::OnBypass()
{
  // TODO: Add your control notification handler code here
  int state = m_BypassButton.GetState() & 0x3;

  g_DSPManager->SetBypassSection(state,kBypassAllpass);

}

BOOL CAllpassPage::OnInitDialog()
{
  CPropertyPage::OnInitDialog();

  // TODO: Add extra initialization here
  m_FrequencySlider.SetRange(0,CONTROL_RANGE);
  m_FrequencySlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kAllpassFreq));
  m_FrequencySlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_QSlider.SetRange(0,CONTROL_RANGE);
  m_QSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kAllpassQ));
  m_QSlider.SetTicFreq((CONTROL_RANGE+1)/16);
```

0012

```
  m_ParentWindow = (CTabDialog *) GetParent()->GetParent();

  return TRUE;  // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CAllpassPage::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
  // TODO: Add your message handler code here and/or call default
  CSliderCtrl *slider = (CSliderCtrl *)pScrollBar;
  int which;

  CPropertyPage::OnVScroll(nSBCode, nPos, pScrollBar);

  Sleep(50);
  if( slider == &m_FrequencySlider )
    which = kAllpassFreq;
  else if( slider == &m_QSlider )
    which = kAllpassQ;
  else
    return;

  g_DSPManager->SetParamValue(CONTROL_RANGE-slider->GetPos(),which);
  SendStringToUI(which);

}

void CAllpassPage::SendStringToUI(int which)
{

    CString str;

    g_DSPManager->GetStringValue(which,str);
    m_ParentWindow->SetStatusString(0,str);

}


void CAllpassPage::OnPaint()
{
  CPaintDC dc(this); // device context for painting

  // TODO: Add your message handler code here
  m_FrequencySlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kAllpassFreq));
  m_QSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kAllpassQ));
  m_BypassButton.SetCheck(g_DSPManager->GetBypassSection(kBypassAllpass));

  // Do not call CPropertyPage::OnPaint() for painting messages
}
```

0013

```
// DSPComm.h: interface for the DSPComm class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_DSPCOMM_H__33F9EF05_F6EF_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_DSPCOMM_H__33F9EF05_F6EF_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class DSPComm
{
public:
  virtual BOOL IsTransmitting();
  virtual long GetDetectState(void);
  virtual BOOL CheckState(void);
  virtual long SendDSPMemory(char *, long);
  virtual long SendDSPWord(long);
  DSPComm();
  virtual ~DSPComm();

};

#endif // !defined(AFX_DSPCOMM_H__33F9EF05_F6EF_11D2_96EE_006097CDB9E2__INCLUDED_)
```

```
// DSPComm.cpp: implementation of the DSPComm class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "sa.h"
#include "DSPComm.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

DSPComm::DSPComm()
{

}

DSPComm::~DSPComm()
{

}

long DSPComm::SendDSPWord(long)
{

  return( 0L );
}

long DSPComm::SendDSPMemory(char *data, long len)
{

  return( 0L );
}

BOOL DSPComm::CheckState()
{
  return( true );
}

long DSPComm::GetDetectState()
{

  return( 0L );

}

BOOL DSPComm::IsTransmitting()
{

  return( false );

}
```

0015

```
// DSP56kManager.h: interface for the CDSP56kManager class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_DSP56KMANAGER_H__0CF7E204_D790_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_DSP56KMANAGER_H__0CF7E204_D790_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "DSP56Equates.h"
#include "DSPComm.h"

#define CONTROL_RANGE_L2  12
#define CONTROL_RANGE     ((1L<<CONTROL_RANGE_L2)-1)
#define PG_CONTROL_AMT  (CONTROL_RANGE/64)
#define CONTROL_RANGE_SC  (23-CONTROL_RANGE_L2)
#define DSP_CONTROL_MAX 8388608.

class CDSP56kManager
{

  long  m_DSPReturn;
  long  m_Raw2DSPVals[kUIArraySize];
  long  m_RawVals[kUIArraySize];
  long  m_DSPVals[kDSPArraySize];
  double m_pi;
  DSPComm *m_Comm;

public:
  void GetPureStringValue(int which, CString &str);
  void GetFilterBlob(CStringArray &array);
  virtual BOOL GetBypassSection(int which);
  virtual void SetBypassSection(BOOL value, int which);
  virtual BOOL IsBusy();
  virtual void GetStringValue(int which,CString &str);
  virtual BOOL IsReady(void);
  virtual void SetAnalogInput(long);
  virtual int GetHDCDMode(void);
  virtual void ResetAll(void);
  virtual void SetHDCDGainScale(long);
  virtual void SetHDCDBypass(long);
  virtual void SetDDXCompBypass(long);
  virtual void SetBypass(long value);
  virtual void GetDSPSettings(CStringArray &array);
  virtual void SetDSPSettings(CStringArray &array);
  virtual void SetParamValue(long value, long which);
  virtual long GetParamValue(long which);
  void DownloadDSPCode(void);

  CDSP56kManager(HWND p);
  virtual ~CDSP56kManager();

private:
  void SetNotchQ(long value,long which,long freqwhich);
  void SetNotchFreq(long value,long which);
  void SetShelvFreq(long value,long which);
  void SetBoostCut(long value,long which);
  void SetRawValue(long value, long which);
  double DoConvertFreqRange(double in,double top,double bottom);
  double ConvertFreqRange(int which);
  double ConvertQRange(long val);
  double ConvertBoostCutRange(long val);
  virtual void SendDSPValue(long which);

  int ChecksumSREC(char *lineptr,int N);
  void GetSRecordAddressRange(char *s, long *start, long *end, char **data);

protected:
  virtual void SetDelay(long value);
  void SetLoCutoff(void);
```

0016

```
  void SetHiCutoff(void);
  void SetHiCutoff2(void);

  virtual void SetHiCutoff2Q(long value);
  virtual void SetHiCutoff2Freq(long value);
  virtual void SetHiCutoffQ(long value);
  virtual void SetHiCutoffFreq(long value);
  virtual void SetLoCutoffQ(long value);
  virtual void SetLoCutoffFreq(long value);
  virtual void SetPreVolume(long value);
  virtual void SetAnalogVolume(long value);

//  virtual void SetCutoff1Freq(long value);
//  virtual void SetCutoff1Q(long value);
//  virtual void SetCutoff1Boost(long value);
//  virtual void SetCutoff2Freq(long value);
//  virtual void SetCutoff2Q(long value);
//  virtual void SetCutoff2Boost(long value);
  virtual void SetShelv1Freq(long value);
  virtual void SetShelv1Boost(long value);
  virtual void SetShelv2Freq(long value);
  virtual void SetShelv2Boost(long value);
  virtual void SetNotch1Cut(long value);
  virtual void SetNotch1Q(long value);
  virtual void SetNotch1Freq(long value);
  virtual void SetNotch2Cut(long value);
  virtual void SetNotch2Q(long value);
  virtual void SetNotch2Freq(long value);
  virtual void SetNotch3Cut(long value);
  virtual void SetNotch3Q(long value);
  virtual void SetNotch3Freq(long value);
  virtual void SetNotch4Cut(long value);
  virtual void SetNotch4Q(long value);
  virtual void SetNotch4Freq(long value);
  virtual void SetNotch5Q(long value);
  virtual void SetNotch5Boost(long value);
  virtual void SetNotch6Cut(long value);
  virtual void SetNotch6Q(long value);
  virtual void SetNotch6Freq(long value);
  virtual void SetNotch7Cut(long value);
  virtual void SetNotch7Q(long value);
  virtual void SetNotch7Freq(long value);
  virtual void SetNotch9Freq(long value);

  virtual void SetMainVolume(long value);

};

extern  CDSP56kManager *g_DSPManager;

#endif // !defined(AFX_DSP56KMANAGER_H__0CF7E204_D790_11D2_96EE_006097CDB9E2__INCLUDED_)
```

0017 A

```
// DSP56kManager.cpp: implementation of the CDSP56kManager class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "sa.h"
#include "DSP56kManager.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#if 0
#include "unit_ppi.h"
#include "functs.h"
#endif

#ifdef  SPI
#include "SPIPEMicroComm.h"
#endif
#include "I2CIPortComm.h"


#include "sapmem.h"
#include "SSTParams.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

#if 0
static    int cmd_delay = 1000;
static    int io_delay = 50;
#endif

#define FILE_VERSION  2     // file version
#define GAIN_SCALE 8.

// volume
#define VOLTOP  1.
#define VOLBOT  3.909e-6
#define VOLRANGE  (VOLTOP-VOLBOT)

// hi and lo cutoff and shelving ranges
#define FLCTOPRANGE 0.149           // 1000/22050
#define FLCBOTRANGE 0.001769        // 10/22050
#define FLCRANGE (FLCTOPRANGE-FLCBOTRANGE)

#define FLCTOPRANGE2  4.255           // 2000
#define FLCBOTRANGE2  2.134           // 15

#define FHCTOPRANGE 1           // 22050/22050
#define FHCBOTRANGE 0.63        // 8000/22050
#define FHCBOTRANGE2  0.42        // 4000/22050
#define FHCRANGE (FHCTOPRANGE-FHCBOTRANGE)
#define FHCRANGE2 (FHCTOPRANGE-FHCBOTRANGE2)

#define QLCTOPRANGE 1.0             // 1
#define QLCBOTRANGE 0.003891        // 0.001
#define QLCRANGE (QLCTOPRANGE-(QLCBOTRANGE))

#define SAMPLING_FREQ     44100
#define DELAY_LENGTH      128.

// converts frequency range to log scale
#define FTOPRANGE 1.
#define FBOTRANGE 0.134
#define FRANGE (FTOPRANGE-FBOTRANGE)
#define FTOPRANGE2 0.585    // about 2205 top.
#define FBOTRANGE2  3.531E-3
```

```
// new ranges using actual frequencies
#define FTOPRANGE3  4.431   // 3000 Hz
#define FBOTRANGE3  2.433   // 30 Hz

// hi cutoff filter ranges
#define QHC2BOT   0.74
#define QHC2TOP   2.258
#define FHC2BOT   3.954   // 1000 Hz
#define FHC2TOP   5.255   // 20 kHz

#define FHCTOPRANGE3  5.13            // 15000
#define FHCBOTRANGE3  FHC2BOT


static char *m_ParamNames[kUIArraySize] = { "MainVolume",
    "LoShelfFreq",
    "LoShelfGain",
    "HiShelfFreq",
    "HiShelfGain",
    "Notch1Freq",
    "Notch1Q",
    "Notch1Cut",
    "Notch2Freq",
    "Notch2Q",
    "Notch2Cut",
    "Notch3Freq",
    "Notch3Q",
    "Notch3Cut",
    "Notch4Freq",
    "Notch4Q",
    "Notch4Cut",
    "Notch5Freq",
    "Notch5Q",
    "Notch5Cut",
    "PreVolume",
    "Bypass",
    "LoCutoffFreq",
    "LoCutoffQ",
    "HiCutoffFreq",
    "HiCutoffQ",
    "HDCDBypass",
    "HDCDGainScaleOff",
    "DDXCompBypass",
    "Notch6Freq",
    "Notch6Q",
    "Notch6Cut",
    "Notch7Freq",
    "Notch7Q",
    "Notch7Cut",
    "AnalogIn",
    "AnalogVolume",
    "Delay",
    "AllpassFreq",
    "AllpassQ",
    "BypassMask",
    "HiCutoff2Freq",
    "HiCutoff2Q",
    "Notch8Freq",
    "Notch8Q",
    "Notch8Cut",
    "Notch9Freq",
    "Notch9Q",
    "Notch9Cut",
    "NotchAFreq",
    "NotchAQ",
    "NotchACut",
    "DBNotchWidth" };

////////////////////////////////////////////////////////////////////////
// SREC code
////////////////////////////////////////////////////////////////////////
```

0018

```
int CDSP56kManager::ChecksumSREC(char *lineptr,int N)
{

  unsigned char curbyte,sum=0xff;
  for (int x=2;x<N-2;x+=2)
  {
    sscanf(lineptr+x,"%2x",&curbyte);
    sum-=curbyte;
  }

  // last one is checksum
  sscanf(lineptr+x,"%2x",&curbyte);
  return (curbyte==sum);

}

void CDSP56kManager::GetSRecordAddressRange(char *s, long *start, long *end, char **data)
{

  long address,count,soffset;

  *start = -1L;
  *end = -1L;

  if (*s != 'S') // not a srec
        return;

  // Let's check the checksum for this line
  ASSERT ( ChecksumSREC(s,strlen(s)) );
  switch( s[1] )
    {
  case '1':
        soffset=8;
        sscanf(&(s[4]),"%4x",&address);
        break;
  case '2':
        soffset=10;
        sscanf(&(s[4]),"%6x",&address);
        break;
  case '3':
        soffset=12;
        sscanf(&(s[4]),"%8x",&address);
        break;
  default:
    return;
    break;

    }

  count = strlen(s) - soffset - 2;
  *start = address;
  *end = address + (count/6);      // each word is 6 characters
  *data = s + soffset;

}


void CDSP56kManager::DownloadDSPCode()
{

    char **r = pmem;
    int count = PRECORDS;
    long start,end;
    long ts,te;
    char **recs;
    char *srecdata,*td;
    char *data,*dptr,*rdata,c;

    long size,bsize;
    long taddress,recstart,recend;
    long value;
```

0019

```
// get starting and ending addresses
start  = 0x7FFFFFFFL;
end = 0;
recs = r;
for( int i = 0; i < count ; i++ )
{
  GetSRecordAddressRange(*recs++,&ts,&te,&srecdata);
  if( ts < 0L ) continue;
  start = min(start,ts);
  end = max(end,te);
}

size = end - start;
bsize = size * 3;
rdata = (char *) malloc(bsize+6);      // 3 bytes per word plus length and address
ASSERT( data != NULL );
data = rdata + 6;

dptr = data;
for( i = 0 ; i < bsize ; i++ )     // clear memory
  *dptr++ = 0;
taddress = start;
while( taddress < end )
{
  recstart = 0x7FFFFFFFL;
  recs = r;
  for( int i = 0; i < count ; i++ )            // search for next address
  {
    GetSRecordAddressRange(*recs,&ts,&te,&td);
    if( ts >= 0L )
    {
      if( ts >= taddress )
      {
        if( ts < recstart )
        {
          recstart = ts;
          recend = te;
          srecdata = td;
        }
      }
    }
    recs++;
  }
  ASSERT( recstart <= end );
  // copy record
  dptr = data + ((recstart - start)*3);
  while( recstart != recend )
  {
    sscanf(srecdata,"%2x",&value);
    *dptr++ = (char ) value;
    srecdata += 2;
    sscanf(srecdata,"%2x",&value);
    *dptr++ = (char ) value;
    srecdata += 2;
    sscanf(srecdata,"%2x",&value);
    *dptr++ = (char ) value;
    srecdata += 2;
    recstart++;
  }
  taddress = recend;
}

// copy size and address
dptr = rdata;
c = (size >> 16) & 0xFF;
*dptr++ = c;
c = (size >> 8) & 0xFF;
*dptr++ = c;
c = size & 0xFF;
*dptr++ = c;
c = (start >> 16) & 0xFF;
```

0020

```
        *dptr++ = c;
        c = (start >> 8) & 0xFF;
        *dptr++ = c;
        c = start & 0xFF;
        *dptr++ = c;

        // send data to DSP
        //m_Comm->SendDSPWord(size);
        //m_Comm->SendDSPWord(start);
        m_Comm->SendDSPMemory(rdata,bsize+6);

        free( rdata );

    }

    ////////////////////////////////////////////////////////////////////////
    // Construction/Destruction
    ////////////////////////////////////////////////////////////////////////

    CDSP56kManager::CDSP56kManager(HWND p)
    {

    #if 0
      if( load_dll()==false )
      {
        MessageBox(NULL,"UNIT_PPI not found in the system directory. Fix this and restart application.","Danger,
    anger Will Robinson", MB_OK | MB_ICONSTOP );
      }
      else
      {
        init_port_spi(1);
        set_cmd_delay_cnt_value(cmd_delay);
        set_io_delay_cnt_value(io_delay);
      }
    #endif
      m_pi = acos(-1.);

      m_Comm = new I2CIPortComm(p);

    //  DownloadDSPCode();


      int i;
      for( i = 0 ; i < kUIArraySize ; i++ )
      {
        m_Raw2DSPVals[i] = i;
      }

      m_Raw2DSPVals[kNotch6Freq] = kDSPNotch6Freq;
      m_Raw2DSPVals[kNotch6Q] = kDSPNotch6Q;
      m_Raw2DSPVals[kNotch6Cut] = kDSPNotch6Cut;
      m_Raw2DSPVals[kNotch7Freq] = kDSPNotch7Freq;
      m_Raw2DSPVals[kNotch7Q] = kDSPNotch7Q;
      m_Raw2DSPVals[kNotch7Cut] = kDSPNotch7Cut;
      m_Raw2DSPVals[kDelay] = kDSPDelay;
      m_Raw2DSPVals[kAllpassFreq] = kDSPAllpassFreq;
      m_Raw2DSPVals[kAllpassQ] = kDSPAllpassQ;

      m_Raw2DSPVals[kNotch8Freq] = kDSPNotch8Freq;
      m_Raw2DSPVals[kNotch8Q] = kDSPNotch8Q;
      m_Raw2DSPVals[kNotch8Cut] = kDSPNotch8Cut;
      m_Raw2DSPVals[kNotch9Freq] = kDSPNotch9Freq;
      m_Raw2DSPVals[kNotch9Q] = kDSPNotch9Q;
      m_Raw2DSPVals[kNotch9Cut] = kDSPNotch9Cut;
      m_Raw2DSPVals[kNotchAFreq] = kDSPNotchAFreq;
      m_Raw2DSPVals[kNotchAQ] = kDSPNotchAQ;
      m_Raw2DSPVals[kNotchACut] = kDSPNotchACut;

      ResetAll();

    }
```

0021

```
CDSP56kManager::~CDSP56kManager()
{

  if( m_Comm )
    delete m_Comm;

}

void CDSP56kManager::SetMainVolume(long value)
{

  double fvalue;

  if( value >= 0 )
    SetRawValue(value,kMainVolume);
  if( m_RawVals[kMainVolume] == DSP_CONTROL_MAX )
    m_DSPVals[kDSPMainVolume] = -m_RawVals[kMainVolume];
  else if( m_RawVals[kMainVolume] == 0 )
    m_DSPVals[kDSPMainVolume] = 0;
  else
    {
      fvalue = m_RawVals[kMainVolume];
      fvalue /= DSP_CONTROL_MAX;     // normalize
      fvalue *= VOLRANGE;
      fvalue += VOLBOT;
      fvalue = (pow(10.,fvalue) - 1.)/(10.-1.);
      fvalue *= DSP_CONTROL_MAX;
      m_DSPVals[kDSPMainVolume] = (long) -fvalue;
    }
  SendDSPValue(kDSPMainVolume);

}

void CDSP56kManager::SendDSPValue(long which)
{

  long value = m_DSPVals[which];

#if 0

  // transmitted value has parameter value in top 16 bits and parameter identifier in 8 LSBs
  value = (value & 0xFFFF00) | (which & 0xFF);
//  m_DSPReturn = spi_xchng24(value);
  m_DSPReturn = m_Comm->SendDSPWord(value);
#endif

  // changed protocol around to send index followed by data
  char  buffer[6];

  buffer[0] = (which >> 16) & 0xFF;
  buffer[1] = (which >> 8) & 0xFF;
  buffer[2] = which & 0xFF;
  buffer[3] = (value >> 16) & 0xFF;
  buffer[4] = (value >> 8) & 0xFF;
  buffer[5] = value & 0xFF;
  m_DSPReturn = m_Comm->SendDSPMemory(buffer,6);

}


void CDSP56kManager::SetNotchFreq(long ivalue,long which)
{

  double value;
  int dspindex = m_Raw2DSPVals[which];

  if( ivalue >= 0 )
    SetRawValue(ivalue,which);
  value = ConvertFreqRange(which);
  value = - cos(m_pi*value);
  value *= DSP_CONTROL_MAX;
  m_DSPVals[dspindex] = (long) value;
```

0022

```
  SendDSPValue(dspindex);
  SetNotchQ(-1L,which+1,which);

}

#ifdef  LEGACY
void CDSP56kManager::SetShelvFreq(long ivalue,long which)
{

  double value,wc;
  int dspindex = m_Raw2DSPVals[which];

  if( ivalue >= 0 )
    SetRawValue(ivalue,which);
  value = ConvertFreqRange(which);
  wc = m_pi*value;
  value = (tan(wc/2)-1)/(tan(wc/2)+1);
  value *= DSP_CONTROL_MAX;
  m_DSPVals[dspindex] = (long) value;
  SendDSPValue(dspindex);

}
#endif

void CDSP56kManager::SetNotchQ(long ivalue,long which,long rawfreq)
{

  double beta,tbeta,q,wc;
  int dspindex = m_Raw2DSPVals[which];

  if( ivalue >= 0 )
    SetRawValue(ivalue,which);
  wc = ConvertFreqRange(rawfreq);
  wc = m_pi*wc;
  q = ConvertQRange(m_RawVals[which]);
  tbeta = wc/(2*q);
  if( tbeta > m_pi/4 )
    tbeta = m_pi/4;
  beta = (1.-tan(tbeta))/(1.+tan(tbeta));
  beta *= DSP_CONTROL_MAX;
  m_DSPVals[dspindex] = (long) beta;
  SendDSPValue(dspindex);

}

void CDSP56kManager::SetBoostCut(long ivalue,long which)
{

  double value;
  int dspindex = m_Raw2DSPVals[which];

  if( ivalue >= 0 )
    SetRawValue(ivalue,which);
  value = ConvertBoostCutRange(m_RawVals[which]);
  value = pow(10.,value);
  value /= GAIN_SCALE;
  value *= DSP_CONTROL_MAX;
  m_DSPVals[dspindex] = (long) value;
  SendDSPValue(dspindex);

}

void CDSP56kManager::SetShelv1Freq(long ivalue)
{

//  SetShelvFreq(value,kLoShelfFreq);
  double value,wc;

  if( ivalue >= 0 )
    SetRawValue(ivalue,kLoShelfFreq);
  value = m_RawVals[kLoShelfFreq];
  value /= DSP_CONTROL_MAX;    // normalize
```

```
  value *= FLCRANGE;
  value += FLCBOTRANGE;
  value = (pow(10.,value) - 1.)/(10.-1.);
  wc = m_pi*value;
  value = (tan(wc/2)-1)/(tan(wc/2)+1);
  value *= DSP_CONTROL_MAX;
  m_DSPVals[kLoShelfFreq] = (long) value;
  SendDSPValue(kLoShelfFreq);

}

void CDSP56kManager::SetShelv1Boost(long value)
{

  SetBoostCut(value,kLoShelfGain);

}

void CDSP56kManager::SetShelv2Freq(long ivalue)
{

//  SetShelvFreq(value,kHiShelfFreq);
  double value,wc;

  if( ivalue >= 0 )
    SetRawValue(ivalue,kHiShelfFreq);
  value = m_RawVals[kHiShelfFreq];
  value /= DSP_CONTROL_MAX;   // normalize
  value *= FHCRANGE2;
  value += FHCBOTRANGE2;
  value = (pow(10.,value) - 1.)/(10.-1.);
  wc = m_pi*value;
  value = (tan(wc/2)-1)/(tan(wc/2)+1);
  value *= DSP_CONTROL_MAX;
  m_DSPVals[kHiShelfFreq] = (long) value;
  SendDSPValue(kHiShelfFreq);

}

void CDSP56kManager::SetShelv2Boost(long value)
{

  SetBoostCut(value,kHiShelfGain);

}

void CDSP56kManager::SetNotch1Freq(long value)
{

  SetNotchFreq(value,kNotch1Freq);

}

void CDSP56kManager::SetNotch1Q(long value)
{

  SetNotchQ(value,kNotch1Q,kNotch1Freq);

}

void CDSP56kManager::SetNotch1Cut(long value)
{

  SetBoostCut(value,kNotch1Cut);

}

void CDSP56kManager::SetNotch2Freq(long value)
{

  SetNotchFreq(value,kNotch2Freq);
```

```
}

void CDSP56kManager::SetNotch2Q(long value)
{

  SetNotchQ(value,kNotch2Q,kNotch2Freq);

}

void CDSP56kManager::SetNotch2Cut(long value)
{

  SetBoostCut(value,kNotch2Cut);

}


void CDSP56kManager::SetNotch9Freq(long value)
{

  double q,f,nf;

  SetNotchFreq(value,kNotch9Freq);
  f = ConvertFreqRange(kNotch9Freq);
  q = ConvertQRange(m_RawVals[kNotch9Q]);
  nf = f - (f / (5.*q));
  if( nf < 0. )
    nf = 0.;
  nf = pow(nf,0.5);
  nf = log10((nf * (10.-1.)) + 1.);
  nf -= FBOTRANGE;
  nf /= (FTOPRANGE-FBOTRANGE);
  nf *= DSP_CONTROL_MAX;     // normalize
  m_RawVals[kNotch8Freq] = (long) nf;
  nf = f + (f / (5.*q));
  if( nf > 1. )
    nf = 1.;
  nf = pow(nf,0.5);
  nf = log10((nf * (10.-1.)) + 1.);
  nf -= FBOTRANGE;
  nf /= (FTOPRANGE-FBOTRANGE);
  nf *= DSP_CONTROL_MAX;     // normalize
  m_RawVals[kNotchAFreq] = (long) nf;
  SetNotchFreq(-1L,kNotch8Freq);
  SetNotchFreq(-1L,kNotchAFreq);

}

void CDSP56kManager::SetNotch3Freq(long value)
{

  SetNotchFreq(value,kNotch3Freq);
  m_RawVals[kNotch5Freq] = (m_RawVals[kNotch4Freq] + m_RawVals[kNotch3Freq])/2;
  SetNotchFreq(-1L,kNotch5Freq);

}

void CDSP56kManager::SetNotch3Q(long value)
{

  SetNotchQ(value,kNotch3Q,kNotch3Freq);


}

void CDSP56kManager::SetNotch3Cut(long value)
{

  SetBoostCut(value,kNotch3Cut);

}
```

```
void CDSP56kManager::SetNotch4Freq(long value)
{

  SetNotchFreq(value,kNotch4Freq);
  m_RawVals[kNotch5Freq] = (m_RawVals[kNotch4Freq] + m_RawVals[kNotch3Freq])/2;
  SetNotchFreq(-1L,kNotch5Freq);

}

void CDSP56kManager::SetNotch4Q(long value)
{

  SetNotchQ(value,kNotch4Q,kNotch4Freq);

}

void CDSP56kManager::SetNotch4Cut(long value)
{

  SetBoostCut(value,kNotch4Cut);

}

void CDSP56kManager::SetNotch5Q(long value)
{

  SetNotchQ(value,kNotch5Q,kNotch5Freq);

}

void CDSP56kManager::SetNotch5Boost(long value)
{

  SetBoostCut(value,kNotch5Cut);

}


double CDSP56kManager::DoConvertFreqRange(double val,double top, double bottom)
{

  val /= DSP_CONTROL_MAX;   // normalize
  val *= top - bottom;
  val += bottom;
  val = (pow(10.,val) - 1.)/(10.-1.);
  return( val );

}

double CDSP56kManager::ConvertFreqRange(int which)
{

  double fval;

  switch( which )
    {
      case kNotch1Freq:
      case kNotch2Freq:
        fval = DoConvertFreqRange(m_RawVals[which],FTOPRANGE3,FBOTRANGE3);    // top frequency is 0.1 * Nyquis
t
        fval /= SAMPLING_FREQ/2;
        break;
      default:
        fval = DoConvertFreqRange(m_RawVals[which],FTOPRANGE,FBOTRANGE);
        // square log scale
        fval = fval * fval;
        break;
    }
  return( fval );

}
```

0026

```
// converts Q range to log scale
#define QTOPRANGE 2.258
#define QBOTRANGE 0.037
#define QRANGE (QTOPRANGE-QBOTRANGE)

double CDSP56kManager::ConvertQRange(long ival)
{

  double val;

  val = ival;
  val /= DSP_CONTROL_MAX;   // normalize
  val *= QRANGE;
  val += QBOTRANGE;
  val = (pow(10.,val) - 1.)/(10.-1.);
  return( val );

}

// converts input value to log gain value
//#define LOG8  0.9031
#define LOGGAINSCALE  (log10(GAIN_SCALE))
double CDSP56kManager::ConvertBoostCutRange(long ival)
{

  double val;

  val = ival;
  val /= DSP_CONTROL_MAX;   // normalize
  val = (2.*val*LOGGAINSCALE) - LOGGAINSCALE;
  return( val );

}

void CDSP56kManager::SetRawValue(long value,long which)
{

  if( value )
    value += 1;
  value <<= CONTROL_RANGE_SC;
  m_RawVals[which] = value;

}

long CDSP56kManager::GetParamValue(long which)
{

  long value;

  if( which >= kUIArraySize )
    value = 0x400000L >> CONTROL_RANGE_SC;
  else
    value = m_RawVals[which] >> CONTROL_RANGE_SC;
  value--;
  if( value < 0 )
    value = 0;
  return( value );

}

void CDSP56kManager::SetParamValue(long value, long which)
{

    switch( which )
    {
    case kHiCutoff2Freq:
      SetHiCutoff2Freq(value);
      break;
    case kHiCutoff2Q:
      SetHiCutoff2Q(value);
      break;
    case kHiCutoffFreq:
```

```
    SetHiCutoffFreq(value);
    break;
  case kHiCutoffQ:
    SetHiCutoffQ(value);
    break;
  case kLoCutoffFreq:
    SetLoCutoffFreq(value);
    break;
  case kLoCutoffQ:
    SetLoCutoffQ(value);
    break;
  case kBypass:
    SetBypass(value);
    break;
  case kDDXCompBypass:
    SetDDXCompBypass(value);
    break;
  case kHDCDBypass:
    SetHDCDBypass(value);
    break;
  case kHDCDGainScaleOff:
    SetHDCDGainScale(value);
    break;
  case kAnalogVolume:
    SetAnalogVolume(value);
    break;
  case kMainVolume:
    SetMainVolume(value);
    break;
  case kPreVolume:
    SetPreVolume(value);
    break;
  case kLoShelfFreq:
    SetShelv1Freq(value);
    break;
  case kHiShelfFreq:
    SetShelv2Freq(value);
    break;
  case kNotch8Cut:
    SetBoostCut(value,which);
    SetBoostCut(value,kNotchACut);      // same value for both notches
    break;
  case kNotch1Cut:
  case kNotch2Cut:
  case kNotch3Cut:
  case kNotch4Cut:
  case kNotch5Cut:
  case kNotch6Cut:
  case kNotch7Cut:
  case kLoShelfGain:
  case kHiShelfGain:
    SetBoostCut(value,which);
    break;
  case kNotch9Cut:
    m_DSPVals[kDSPNotch9Cut] = 0;      // fixed notch
    SendDSPValue(kDSPNotch9Cut);
    break;
  case kNotch3Freq:
    SetNotch3Freq(value);
    break;
  case kNotch4Freq:
    SetNotch4Freq(value);
    break;
  case kNotch9Freq:
    SetNotch9Freq(value);
    break;
  case kNotch1Freq:
  case kNotch2Freq:
  case kNotch5Freq:
  case kNotch6Freq:
  case kNotch7Freq:
  case kAllpassFreq:
```

0028

```
      SetNotchFreq(value,which);
      break;
    case kNotch8Q:
      SetNotchQ(value,which,which-1);
      SetNotchQ(value,kNotchAQ,kNotchAQ-1);    // same value for both notches
      break;
    case kNotch9Q:
      SetNotchQ(value,which,which-1);
      SetNotch9Freq(-1);
      break;
    case kNotch1Q:
    case kNotch2Q:
    case kNotch3Q:
    case kNotch4Q:
    case kNotch5Q:
    case kNotch6Q:
    case kNotch7Q:
    case kAllpassQ:
      SetNotchQ(value,which,which-1);
      break;
    case kDelay:
      SetDelay(value);
      break;
//    case kDBNotchWidth:
//      SetDelay(value);
      break;
    case kNotch8Freq:
    case kNotchAFreq:
    case kNotchACut:
    case kNotchAQ:
      break;
    }

}


void CDSP56kManager::SetDSPSettings(CStringArray & array)
{

  int i;
  CString string;
  int index;
  long value;
  int fileversion = 0;
  int j = 0;
  double fvalue;
  const char *sptr;

  // reset all parameters
  ResetAll();

  // first check for comment at head
  string = array.GetAt(0);
  sptr = string;
  if( strcmp(sptr,"# VERSION") == 0 )
    {
      string = array.GetAt(1);
      sptr = string;
      sscanf(sptr,"%d",&fileversion);
      j = 2;
    }
  for( i = j ; i < array.GetSize() ; i++ )
    {
        string = array.GetAt(i);
        sptr = string;
        if( sptr[0] == '#' ) continue;      // skip comments
        sscanf(sptr,"%d\t0x%x",&index,&value);
        if( index >= kUIArraySize ) continue;
        // Add any special treatment of parameters here
        if( fileversion == 0L )
        {
          switch( index )
```

```
    {
    case kNotch1Freq:
    case kNotch2Freq:
      fvalue = DoConvertFreqRange(value,FTOPRANGE,FBOTRANGE);
      fvalue = fvalue * fvalue;
      fvalue *= SAMPLING_FREQ/2;
      fvalue = log10((fvalue * (10.-1.)) + 1.);
      fvalue -= FBOTRANGE3;
      fvalue /= (FTOPRANGE3-FBOTRANGE3);
      if( fvalue > 1.0 )
        fvalue = 1.0;
      fvalue *= DSP_CONTROL_MAX;     // normalize
      value = (long) fvalue;
      break;
      // increased low frequency
    case kHiShelfFreq:
      fvalue = DoConvertFreqRange(value,FHCTOPRANGE,FHCBOTRANGE);
      fvalue = log10((fvalue * (10.-1.)) + 1.);
      fvalue -= FHCBOTRANGE2;
      fvalue /= (FHCTOPRANGE-FHCBOTRANGE2);
      if( fvalue > 1.0 )
        fvalue = 1.0;
      fvalue *= DSP_CONTROL_MAX;     // normalize
      value = (long) fvalue;
      break;
    case kHiCutoffFreq:
      fvalue = DoConvertFreqRange(value,FHCTOPRANGE,FHCBOTRANGE);
      fvalue *= SAMPLING_FREQ/2;
      fvalue = log10((fvalue * (10.-1.)) + 1.);
      fvalue -= FHCBOTRANGE3;
      fvalue /= (FHCTOPRANGE3-FHCBOTRANGE3);
      if( fvalue > 1.0 )
        fvalue = 1.0;
      fvalue *= DSP_CONTROL_MAX;     // normalize
      value = (long) fvalue;
      break;
    }
  }
  else if( fileversion == 1L )
  {
    switch( index )
    {
    case kNotch1Freq:
    case kNotch2Freq:
      fvalue = DoConvertFreqRange(value,FTOPRANGE2,FBOTRANGE2);
      fvalue *= SAMPLING_FREQ/2;
      fvalue = log10((fvalue * (10.-1.)) + 1.);
      fvalue -= FBOTRANGE3;
      fvalue /= (FTOPRANGE3-FBOTRANGE3);
      if( fvalue > 1.0 )
        fvalue = 1.0;
      fvalue *= DSP_CONTROL_MAX;     // normalize
      value = (long) fvalue;
      break;
    case kLoCutoffFreq:
      fvalue = DoConvertFreqRange(value,FLCTOPRANGE,FLCBOTRANGE);
      fvalue *= SAMPLING_FREQ/2;
      fvalue = log10((fvalue * (10.-1.)) + 1.);
      fvalue -= FLCBOTRANGE2;
      fvalue /= (FLCTOPRANGE2-FLCBOTRANGE2);
      if( fvalue > 1.0 )
        fvalue = 1.0;
      fvalue *= DSP_CONTROL_MAX;     // normalize
      value = (long) fvalue;
      break;
    case kHiCutoff2Freq:
      fvalue = DoConvertFreqRange(value,FTOPRANGE,FBOTRANGE);
      fvalue *= SAMPLING_FREQ/2;
      fvalue = log10((fvalue * (10.-1.)) + 1.);
      fvalue -= FHC2BOT;
      fvalue /= (FHC2TOP-FHC2BOT);
      if( fvalue > 1.0 )
```

0030

```
            fvalue = 1.0;
          fvalue *= DSP_CONTROL_MAX;     // normalize
          value = (long) fvalue;
          break;
        case kHiCutoffFreq:
          fvalue = DoConvertFreqRange(value,FHCTOPRANGE,FHCBOTRANGE2);
          fvalue *= SAMPLING_FREQ/2;
          fvalue = log10((fvalue * (10.-1.)) + 1.);
          fvalue -= FHCBOTRANGE3;
          fvalue /= (FHCTOPRANGE3-FHCBOTRANGE3);
          if( fvalue > 1.0 )
            fvalue = 1.0;
          fvalue *= DSP_CONTROL_MAX;     // normalize
          value = (long) fvalue;
          break;
        }
      }
      m_RawVals[index] = value;
  }
  // tell DSP of changes
  for( i = 0 ; i < kUIArraySize ; i++ )
    {
    SetParamValue(-1,i);;
    }
  m_DSPVals[kDSPBypassMask] = m_RawVals[kBypassMask];
  SendDSPValue(kDSPBypassMask);

}

void CDSP56kManager::GetDSPSettings(CStringArray & array)
{

  char s[1000];
  int i,index;
  int j = 0;
  CString cstr;

  // write header (so far only version number)
  cstr = "# VERSION";
  array.SetAtGrow(j++, cstr);
  sprintf(s,"%d",FILE_VERSION);
  array.SetAtGrow(j++, s);
  cstr = "# DATA";
  array.SetAtGrow(j++, cstr);

  for( i = 0 ; i < kUIArraySize ; i++ )
    {
      index = kUIArraySize-1-i;
      if( index == kBypass ) continue;
      if( index == kHDCDBypass ) continue;
      if( index == kHDCDGainScaleOff ) continue;
      if( index == kAnalogIn ) continue;
      // MM 7/14/99 Added value in human readable form as comment
      GetStringValue(index,cstr);
      sprintf(s,"# %s %s",m_ParamNames[index],cstr);
      array.SetAtGrow(j++, s);
      sprintf(s,"%d\t0x%x",index,m_RawVals[index]);
      array.SetAtGrow(j++, s);
    }

}

void CDSP56kManager::GetFilterBlob(CStringArray & array)
{

  char s[1000];
  int param;
  int j = 0;
  CString cstr;
  float v;

  // write version number
```

```
//  cstr = "# VERSION";
//  array.SetAtGrow(j++, cstr);
  sprintf(s,"%d",BLOB_FILE_VERSION);
  array.SetAtGrow(j++, s);

  // output delay
  param = kDelay;
  GetPureStringValue(param,cstr);
  sscanf(cstr,"%f",&v);
  if( v != 0. )
  {
//  cstr = m_ParamNames[param];
//  array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_DELAY);
    array.SetAtGrow(j++, s);
    cstr = "1";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(param,cstr);
    array.SetAtGrow(j++, cstr);
  }

  // output pre-gain
  param = kPreVolume;
//  cstr = m_ParamNames[param];
//  array.SetAtGrow(j++, cstr);
  sprintf(s,"%d",BLOCK_GAIN);
  array.SetAtGrow(j++, s);
  cstr = "1";
  array.SetAtGrow(j++, cstr);
  GetPureStringValue(param,cstr);
  array.SetAtGrow(j++, cstr);

  // lo-shelf
  if( !GetBypassSection(kBypassLoshelf) )
  {
//    cstr = "Low-Shelf";
//    array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_LS);
    array.SetAtGrow(j++, s);
    cstr = "2";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kLoShelfFreq,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kLoShelfGain,cstr);
    array.SetAtGrow(j++, cstr);
  }

  // hi-shelf
  if( !GetBypassSection(kBypassHiShelf) )
  {
//    cstr = "High-Shelf";
//    array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_HS);
    array.SetAtGrow(j++, s);
    cstr = "2";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kHiShelfFreq,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kHiShelfGain,cstr);
    array.SetAtGrow(j++, cstr);
  }

  // low peaking cutoff
  if( !GetBypassSection(kBypassHipass) )
  {
//    cstr = "Low Cutoff";
//    array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_HP);
    array.SetAtGrow(j++, s);
    cstr = "2";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kLoCutoffFreq,cstr);
```

```
        array.SetAtGrow(j++, cstr);
        GetPureStringValue(kLoCutoffQ,cstr);
        array.SetAtGrow(j++, cstr);
      }

      // high peaking cutoff #1
      if( !GetBypassSection(kBypassLopass) )
      {
    //    cstr = "High Cutoff";
    //    array.SetAtGrow(j++, cstr);
        sprintf(s,"%d",BLOCK_LP);
        array.SetAtGrow(j++, s);
        cstr = "2";
        array.SetAtGrow(j++, cstr);
        GetPureStringValue(kHiCutoffFreq,cstr);
        array.SetAtGrow(j++, cstr);
        GetPureStringValue(kHiCutoffQ,cstr);
        array.SetAtGrow(j++, cstr);
      }

      // high peaking cutoff #2
      if( !GetBypassSection(kBypassNLopass) )
      {
    //    cstr = "High Cutoff #2";
    //    array.SetAtGrow(j++, cstr);
        sprintf(s,"%d",BLOCK_LP2);
        array.SetAtGrow(j++, s);
        cstr = "2";
        array.SetAtGrow(j++, cstr);
        GetPureStringValue(kHiCutoff2Freq,cstr);
        array.SetAtGrow(j++, cstr);
        GetPureStringValue(kHiCutoff2Q,cstr);
        array.SetAtGrow(j++, cstr);
      }

      // resonance comp #1
      if( !GetBypassSection(kBypassNotch1) )
      {
    //    cstr = "Parametric EQ";
    //    array.SetAtGrow(j++, cstr);
        sprintf(s,"%d",BLOCK_PEQ);
        array.SetAtGrow(j++, s);
        cstr = "3";
        array.SetAtGrow(j++, cstr);
        GetPureStringValue(kNotch1Freq,cstr);
        array.SetAtGrow(j++, cstr);
        GetPureStringValue(kNotch1Q,cstr);
        array.SetAtGrow(j++, cstr);
        GetPureStringValue(kNotch1Cut,cstr);
        array.SetAtGrow(j++, cstr);
      }

      // resonance comp #2
      if( !GetBypassSection(kBypassNotch2) )
      {
    //    cstr = "Parametric EQ";
    //    array.SetAtGrow(j++, cstr);
        sprintf(s,"%d",BLOCK_PEQ);
        array.SetAtGrow(j++, s);
        cstr = "3";
        array.SetAtGrow(j++, cstr);
        GetPureStringValue(kNotch2Freq,cstr);
        array.SetAtGrow(j++, cstr);
        GetPureStringValue(kNotch2Q,cstr);
        array.SetAtGrow(j++, cstr);
        GetPureStringValue(kNotch2Cut,cstr);
        array.SetAtGrow(j++, cstr);
      }

      // resonance comp #3
      if( !GetBypassSection(kBypassNotch3) )
      {
```

```
//    cstr = "Parametric EQ";
//    array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_PEQ);
    array.SetAtGrow(j++, s);
    cstr = "3";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch6Freq,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch6Q,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch6Cut,cstr);
    array.SetAtGrow(j++, cstr);
  }

  // resonance comp #4
  if( !GetBypassSection(kBypassNotch4) )
  {
//    cstr = "Parametric EQ";
//    array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_PEQ);
    array.SetAtGrow(j++, s);
    cstr = "3";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch7Freq,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch7Q,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch7Cut,cstr);
    array.SetAtGrow(j++, cstr);
  }

  // conecry
  if( !GetBypassSection(kBypassConecry) )
  {
//    cstr = "Parametric EQ";
//    array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_PEQ);
    array.SetAtGrow(j++, s);
    cstr = "3";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch3Freq,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch3Q,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch3Cut,cstr);
    array.SetAtGrow(j++, cstr);

//    cstr = "Parametric EQ";
//    array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_PEQ);
    array.SetAtGrow(j++, s);
    cstr = "3";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch4Freq,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch4Q,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch4Cut,cstr);
    array.SetAtGrow(j++, cstr);

//    cstr = "Parametric EQ";
//    array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_PEQ);
    array.SetAtGrow(j++, s);
    cstr = "3";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch5Freq,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch5Q,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch5Cut,cstr);
    array.SetAtGrow(j++, cstr);
```

0034

```
  }

  // allpass
  if( !GetBypassSection(kBypassAllpass) )
  {
//    cstr = "Parametric EQ";
//    array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_AP);
    array.SetAtGrow(j++, s);
    cstr = "2";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kAllpassFreq,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kAllpassQ,cstr);
    array.SetAtGrow(j++, cstr);
  }

  // double-tuned notch
  if( !GetBypassSection(kBypassDBNotch) )
  {
//    cstr = "Parametric EQ";
//    array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_PEQ);
    array.SetAtGrow(j++, s);
    cstr = "3";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch8Freq,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch8Q,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch8Cut,cstr);
    array.SetAtGrow(j++, cstr);

//    cstr = "Parametric EQ";
//    array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_PEQ);
    array.SetAtGrow(j++, s);
    cstr = "3";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch9Freq,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch9Q,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotch9Cut,cstr);
    array.SetAtGrow(j++, cstr);

//    cstr = "Parametric EQ";
//    array.SetAtGrow(j++, cstr);
    sprintf(s,"%d",BLOCK_PEQ);
    array.SetAtGrow(j++, s);
    cstr = "3";
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotchAFreq,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotchAQ,cstr);
    array.SetAtGrow(j++, cstr);
    GetPureStringValue(kNotchACut,cstr);
    array.SetAtGrow(j++, cstr);

  }

  // output post-gain
  param = kMainVolume;
//  cstr = m_ParamNames[param];
//  array.SetAtGrow(j++, cstr);
  sprintf(s,"%d",BLOCK_GAIN);
  array.SetAtGrow(j++, s);
  cstr = "1";
  array.SetAtGrow(j++, cstr);
  GetPureStringValue(param,cstr);
  array.SetAtGrow(j++, cstr);
```

```
}

void CDSP56kManager::SetPreVolume(long value)
{

  double fvalue;

  if( value >= 0 )
    SetRawValue(value,kPreVolume);
  if( m_RawVals[kPreVolume] == DSP_CONTROL_MAX )
    m_DSPVals[kDSPPreVolume] = -m_RawVals[kPreVolume];
  else if( m_RawVals[kPreVolume] == 0 )
    m_DSPVals[kDSPPreVolume] = 0;
  else
    {
      fvalue = m_RawVals[kPreVolume];
      fvalue /= DSP_CONTROL_MAX;     // normalize
      fvalue *= VOLRANGE;
      fvalue += VOLBOT;
      fvalue = (pow(10.,fvalue) - 1.)/(10.-1.);
      fvalue *= DSP_CONTROL_MAX;
      m_DSPVals[kDSPPreVolume] = (long) -fvalue;
    }
  SendDSPValue(kDSPPreVolume);

}

void CDSP56kManager::SetAnalogVolume(long value)
{

  double fvalue;

  if( value >= 0 )
    SetRawValue(value,kAnalogVolume);
  if( m_RawVals[kAnalogVolume] == DSP_CONTROL_MAX )
    m_DSPVals[kDSPAnalogVol] = -m_RawVals[kAnalogVolume];
  else if( m_RawVals[kAnalogVolume] == 0 )
    m_DSPVals[kDSPAnalogVol] = 0;
  else
    {
      fvalue = m_RawVals[kAnalogVolume];
      fvalue /= DSP_CONTROL_MAX;     // normalize
      fvalue *= VOLRANGE;
      fvalue += VOLBOT;
      fvalue = (pow(10.,fvalue) - 1.)/(10.-1.);
      fvalue *= DSP_CONTROL_MAX;
      m_DSPVals[kDSPAnalogVol] = (long) -fvalue;
    }
  SendDSPValue(kDSPAnalogVol);

}


void CDSP56kManager::SetBypass(long value)
{

  if( value >= 0 )
    SetRawValue(value,kBypass);
  if( m_RawVals[kBypass] )
    m_DSPVals[kDSPBypass] = 0x7FFFFF;
  else
    m_DSPVals[kDSPBypass] = 0;
  SendDSPValue(kDSPBypass);


}

void CDSP56kManager::SetLoCutoffFreq(long value)
{

  if( value >= 0 )
    SetRawValue(value,kLoCutoffFreq);
```

```
  SetLoCutoff();

}

void CDSP56kManager::SetLoCutoffQ(long value)
{

  if( value >= 0 )
    SetRawValue(value,kLoCutoffQ);
  SetLoCutoff();

}

void CDSP56kManager::SetHiCutoffFreq(long value)
{

  if( value >= 0 )
    SetRawValue(value,kHiCutoffFreq);
  SetHiCutoff();

}

void CDSP56kManager::SetHiCutoffQ(long value)
{

  if( value >= 0 )
    SetRawValue(value,kHiCutoffQ);
  SetHiCutoff();

}

void CDSP56kManager::SetHiCutoff2Freq(long value)
{

  if( value >= 0 )
    SetRawValue(value,kHiCutoff2Freq);
  SetHiCutoff2();

}

void CDSP56kManager::SetHiCutoff2Q(long value)
{

  if( value >= 0 )
    SetRawValue(value,kHiCutoff2Q);
  SetHiCutoff2();

}


void CDSP56kManager::SetHiCutoff()
{

  double f,q;
  double a1,a2,b0;

  q = m_RawVals[kHiCutoffQ];
  q /= DSP_CONTROL_MAX;    // normalize
  q *= QLCRANGE;
  q += QLCBOTRANGE;
  q = (pow(10.,q) - 1.)/(10.-1.);
  if( q >= 1. )
    q = 0.99999;

  /*f = m_RawVals[kHiCutoffFreq];
  f /= DSP_CONTROL_MAX;    // normalize
  f *= FHCRANGE2;
  f += FHCBOTRANGE2;
  f = (pow(10.,f) - 1.)/(10.-1.); */
  f = DoConvertFreqRange(m_RawVals[kHiCutoffFreq],FHCTOPRANGE3,FHCBOTRANGE3);
  f /= SAMPLING_FREQ/2;
  f = 2. * sin( f * m_pi / 2.);
```

0037

```
  a2 = 1. - f * q;
  if( a2 >= 2. )
    a2 = 1.99999;
  else if( a2 <= -2. )
    a2 = -1.99999;
  a1 = -(2. - f * q - f * f);
  if( a1 >= 2. )
    a1 = 1.99999;
  else if( a1 <= -2. )
    a1 = -1.99999;
  b0 = f * f / 2.;
  if( b0 >= 8. )
    b0 = 7.99999;
  else if( b0 <= -8. )
    b0 = -7.99999;

  m_DSPVals[kDSPHiCutoffA2] = (long) (a2 * DSP_CONTROL_MAX/ 2.);
  SendDSPValue(kDSPHiCutoffA2);
  m_DSPVals[kDSPHiCutoffA1] = (long) (a1 * DSP_CONTROL_MAX/2.);
  SendDSPValue(kDSPHiCutoffA1);
  m_DSPVals[kDSPHiCutoffScale] = (long) (b0 * DSP_CONTROL_MAX/ 16.);
  SendDSPValue(kDSPHiCutoffScale);

}

void CDSP56kManager::SetHiCutoff2()
{

  double f,q;
  double a1,a2,b0;
  double gamma,beta,lambda,alpha;

  q = DSP_CONTROL_MAX - m_RawVals[kHiCutoff2Q]; // invert scale
  q /= DSP_CONTROL_MAX;                         // normalize
  q *= (QHC2TOP-QHC2BOT);
  q += QHC2BOT;
  q = (pow(10.,q) - 1.)/(10.-1.);
//  f = ConvertFreqRange(kHiCutoff2Freq);
  f = DoConvertFreqRange(m_RawVals[kHiCutoff2Freq],FHC2TOP,FHC2BOT);
  f /= SAMPLING_FREQ/2;

  gamma = - cos(f * m_pi);
  beta = (1. - sin(f*m_pi/(2*q)))/(1. + sin(f*m_pi/(2*q)));
  if( beta > 2.0 )
    beta = 2.0;
  lambda = 4. * beta * (gamma/(1.+beta));
  if( lambda > 2.0 )
    lambda = 2.0;
  alpha = 1. + beta + lambda;

  a2 = beta;
  a1 = lambda;
  b0 = alpha / 4.;
  if( b0 >= 8. )
    b0 = 7.99999;
  else if( b0 <= -8. )
    b0 = -7.99999;

  m_DSPVals[kDSP2HiCutoffA2] = (long) (a2 * DSP_CONTROL_MAX/ 2.);
  SendDSPValue(kDSP2HiCutoffA2);
  m_DSPVals[kDSP2HiCutoffA1] = (long) (a1 * DSP_CONTROL_MAX/2.);
  SendDSPValue(kDSP2HiCutoffA1);
  m_DSPVals[kDSP2HiCutoffScale] = (long) (b0 * DSP_CONTROL_MAX/ 16.);
  SendDSPValue(kDSP2HiCutoffScale);

}

void CDSP56kManager::SetLoCutoff()
{

  double f,q,scale_factor;
```

```
  q = m_RawVals[kLoCutoffQ];
  q /= DSP_CONTROL_MAX;    // normalize
  q *= QLCRANGE;
  q += QLCBOTRANGE;
  q = (pow(10.,q) - 1.)/(10.-1.);
  if( q >= 1. )
    q = 0.99999;
  f = m_RawVals[kLoCutoffFreq];
  f /= DSP_CONTROL_MAX;    // normalize
  f *= (FLCTOPRANGE2-FLCBOTRANGE2);
  f += FLCBOTRANGE2;
  f = (pow(10.,f) - 1.)/(10.-1.);
  f /= SAMPLING_FREQ/2;
  f = 2. * sin( f * m_pi / 2.);
  if( f >= 1. )
    f = 0.99999;
  scale_factor = (4. - 2. * f * q - f * f)/4.;

  m_DSPVals[kDSPLoCutoffFc] = (long) (f * DSP_CONTROL_MAX);
  SendDSPValue(kDSPLoCutoffFc);
  m_DSPVals[kDSPLoCutoffQc] = (long) (q * DSP_CONTROL_MAX);
  SendDSPValue(kDSPLoCutoffQc);
  m_DSPVals[kDSPLoCutoffScale] = (long) (-0.5 * scale_factor * DSP_CONTROL_MAX);
  SendDSPValue(kDSPLoCutoffScale);
}


void CDSP56kManager::SetDDXCompBypass(long value)
{

  if( value >= 0 )
    SetRawValue(value,kDDXCompBypass);
  if( m_RawVals[kDDXCompBypass] )
    m_DSPVals[kDSPDDXCompBypass] = 0x7FFFFF;
  else
    m_DSPVals[kDSPDDXCompBypass] = 0;
  SendDSPValue(kDSPDDXCompBypass);


}

void CDSP56kManager::SetHDCDBypass(long value)
{

  if( value >= 0 )
    SetRawValue(value,kHDCDBypass);
  if( m_RawVals[kHDCDBypass] )
    m_DSPVals[kDSPHDCDBypass] = 0x7FFFFF;
  else
    m_DSPVals[kDSPHDCDBypass] = 0;
  SendDSPValue(kDSPHDCDBypass);


}

void CDSP56kManager::SetHDCDGainScale(long value)
{

  if( value >= 0 )
    SetRawValue(value,kHDCDGainScaleOff);
  if( m_RawVals[kHDCDGainScaleOff] )
    m_DSPVals[kDSPHDCDGainScaleOff] = 0x7FFFFF;
  else
    m_DSPVals[kDSPHDCDGainScaleOff] = 0;
  SendDSPValue(kDSPHDCDGainScaleOff);

}

void CDSP56kManager::ResetAll()
{
```

0039

```
  int i;
  for( i = 0 ; i < kUIArraySize ; i++ )
    {
    m_RawVals[i] = 0x400000L;
    }
  for( i = 0 ; i < kDSPArraySize ; i++ )
    {
    m_DSPVals[i] = 0;
    }

  m_RawVals[kAnalogVolume] = 0x800000L;
  m_RawVals[kMainVolume] = 0x800000L;
  m_RawVals[kPreVolume] = 0x800000L;
  m_RawVals[kBypass] = 0x000000L;
  m_RawVals[kHDCDBypass] = 0x000000L;
  m_RawVals[kDDXCompBypass] = 0x000000L;
  m_RawVals[kHDCDGainScaleOff] = 0x000000L;
  m_RawVals[kAnalogIn] = 0x000000L;
  m_RawVals[kDelay] = 0x000000L;

  m_RawVals[kHiCutoff2Freq] = 0x600000L;
  m_RawVals[kHiCutoff2Q] = 0x200000L;
  m_RawVals[kHiCutoffFreq] = 0x600000L;
  m_RawVals[kHiCutoffQ] = 0x200000L;
  m_RawVals[kLoCutoffFreq] = 0x40000L;
  m_RawVals[kLoCutoffQ] = 0x200000L;

  m_DSPReturn = 0;

  for( i = 0 ; i < kUIArraySize ; i++ )
  {
  SetParamValue(-1,i);
  }

  m_DSPVals[kDSPBypassMask] = m_RawVals[kBypassMask] = 0L;
  SendDSPValue(kDSPBypassMask);

}

void CDSP56kManager::SetNotch6Freq(long value)
{

  SetNotchFreq(value,kNotch6Freq);

}

void CDSP56kManager::SetNotch6Q(long value)
{

  SetNotchQ(value,kNotch6Q,kNotch6Freq);

}

void CDSP56kManager::SetNotch6Cut(long value)
{

  SetBoostCut(value,kNotch6Cut);

}

void CDSP56kManager::SetNotch7Freq(long value)
{

  SetNotchFreq(value,kNotch7Freq);

}

void CDSP56kManager::SetNotch7Q(long value)
{

  SetNotchQ(value,kNotch7Q,kNotch7Freq);
```

0040

```
}

void CDSP56kManager::SetNotch7Cut(long value)
{

  SetBoostCut(value,kNotch7Cut);

}


int CDSP56kManager::GetHDCDMode()
{

//  SetParamValue(-1,kMainVolume);
//  return( m_DSPReturn );
  return( m_Comm->GetDetectState() );

}

void CDSP56kManager::SetAnalogInput(long value)
{

  if( value >= 0 )
    SetRawValue(value,kAnalogIn);
  if( m_RawVals[kAnalogIn] )
    m_DSPVals[kDSPAnalogIn] = 0x7FFFFF;
  else
    m_DSPVals[kDSPAnalogIn] = 0;
  SendDSPValue(kDSPAnalogIn);

}

BOOL CDSP56kManager::IsReady()
{

  return( m_Comm->CheckState() );

}

void CDSP56kManager::GetPureStringValue(int which, CString &str)
{
    int count;

    GetStringValue(which,str);
    str.TrimLeft();
    count = str.Find(' ');
    if( count >= 0 )
      str = str.Left(count);

}

void CDSP56kManager::GetStringValue(int which,CString &str)
{

    double value;
    char s[100];

    switch( which )
    {
    case kAnalogVolume:
    case kMainVolume:
    case kPreVolume:
      value = m_RawVals[which];
      value /= DSP_CONTROL_MAX;
      value = 20 * log10(value);
      sprintf(s,"%10.2f dB",value);
      str = s;
      break;
    case kHiCutoffFreq:
      /* value = m_RawVals[which];
      value /= DSP_CONTROL_MAX;   // normalize
      value *= FHCRANGE2;
```

```
    value += FHCBOTRANGE2;
    value = (pow(10.,value) - 1.)/(10.-1.);
    value *= SAMPLING_FREQ/2; */
    value = DoConvertFreqRange(m_RawVals[kHiCutoffFreq],FHCTOPRANGE3,FHCBOTRANGE3);
    sprintf(s,"%10.2f Hz",value);
    str = s;
    break;
  case kHiCutoff2Q:
    value = DSP_CONTROL_MAX - m_RawVals[kHiCutoff2Q];
    value /= DSP_CONTROL_MAX;   // normalize
    value *= QHC2TOP-QHC2BOT;
    value += QHC2BOT;
    value = (pow(10.,value) - 1.)/(10.-1.);
    sprintf(s,"%10.2f",value);
    str = s;
    break;
  case kHiCutoffQ:
    value = m_RawVals[kHiCutoffQ];
    value /= DSP_CONTROL_MAX;   // normalize
    value *= QLCRANGE;
    value += QLCBOTRANGE;
    value = (pow(10.,value) - 1.)/(10.-1.);
    if( value >= 1. )
      value = 0.99999;
    sprintf(s,"%10.2f",value);
    str = s;
    break;
  case kLoCutoffFreq:
    value = m_RawVals[kLoCutoffFreq];
    value /= DSP_CONTROL_MAX;   // normalize
    value *= (FLCTOPRANGE2-FLCBOTRANGE2);
    value += FLCBOTRANGE2;
    value = (pow(10.,value) - 1.)/(10.-1.);
    sprintf(s,"%10.2f Hz",value);
    str = s;
    break;
  case kLoCutoffQ:
    value = m_RawVals[kLoCutoffQ];
    value /= DSP_CONTROL_MAX;   // normalize
    value *= QLCRANGE;
    value += QLCBOTRANGE;
    value = (pow(10.,value) - 1.)/(10.-1.);
    if( value >= 1. )
      value = 0.99999;
    sprintf(s,"%10.2f",value);
    str = s;
    break;
  case kLoShelfFreq:
    value = m_RawVals[kLoShelfFreq];
    value /= DSP_CONTROL_MAX;   // normalize
    value *= FLCRANGE;
    value += FLCBOTRANGE;
    value = (pow(10.,value) - 1.)/(10.-1.);
    value *= SAMPLING_FREQ/2;
    sprintf(s,"%10.2f Hz",value);
    str = s;
    break;
  case kHiShelfFreq:
    value = m_RawVals[kHiShelfFreq];
    value /= DSP_CONTROL_MAX;   // normalize
    value *= FHCRANGE2;
    value += FHCBOTRANGE2;
    value = (pow(10.,value) - 1.)/(10.-1.);
    value *= SAMPLING_FREQ/2;
    sprintf(s,"%10.2f Hz",value);
    str = s;
    break;
  case kNotch1Cut:
  case kNotch2Cut:
  case kNotch3Cut:
  case kNotch4Cut:
  case kNotch5Cut:
```

```
    case kNotch6Cut:
    case kNotch7Cut:
    case kNotch8Cut:
    case kNotch9Cut:
    case kNotchACut:
    case kLoShelfGain:
    case kHiShelfGain:
      value = 20 * ConvertBoostCutRange(m_RawVals[which]);
      sprintf(s,"%10.2f dB",value);
      str = s;
      break;
    case kNotch1Freq:
    case kNotch2Freq:
    case kNotch3Freq:
    case kNotch4Freq:
    case kNotch5Freq:
    case kNotch6Freq:
    case kNotch7Freq:
    case kNotch8Freq:
    case kNotch9Freq:
    case kNotchAFreq:
    case kAllpassFreq:
      value = ConvertFreqRange(which);
      value *= SAMPLING_FREQ/2;
      sprintf(s,"%10.2f Hz",value);
      str = s;
      break;
    case kHiCutoff2Freq:
      value = DoConvertFreqRange(m_RawVals[kHiCutoff2Freq],FHC2TOP,FHC2BOT);
      sprintf(s,"%10.2f Hz",value);
      str = s;
      break;
    case kNotch1Q:
    case kNotch2Q:
    case kNotch3Q:
    case kNotch4Q:
    case kNotch5Q:
    case kNotch6Q:
    case kNotch7Q:
    case kNotch8Q:
    case kNotch9Q:
    case kNotchAQ:
    case kAllpassQ:
      value = ConvertQRange(m_RawVals[which]);
      sprintf(s,"%10.2f",value);
      str = s;
      break;
    case kDelay:
      value = m_RawVals[which];
      value /= DSP_CONTROL_MAX;
      value *= DELAY_LENGTH;
      value /= SAMPLING_FREQ;
      value *= 1000.;
      sprintf(s,"%10.2f mS",value);
      str = s;
      break;
    default:
      str = "fixed";
      break;
    }

}

BOOL CDSP56kManager::IsBusy()
{

    return( m_Comm->IsTransmitting() );

}

void CDSP56kManager::SetBypassSection(BOOL value, int which)
{
```

0043

```
  long mask = 1L << which;

  if( value )
    {
      m_RawVals[kBypassMask] |= mask;
    }
  else
    {
      mask = ~mask;
      m_RawVals[kBypassMask] &= mask;
    }
  m_DSPVals[kDSPBypassMask] = m_RawVals[kBypassMask];
  SendDSPValue(kDSPBypassMask);

}

BOOL CDSP56kManager::GetBypassSection(int which)
{

  long mask = 1L << which;

  return( (m_RawVals[kBypassMask] & mask) != 0 );

}

void CDSP56kManager::SetDelay(long value)
{

  if( value >= 0 )
    SetRawValue(value,kDelay);
  m_DSPVals[kDSPDelay] = m_RawVals[kDelay];
  SendDSPValue(kDSPDelay);


}
```

```
// UI IDs
enum {
    kMainVolume,
    kLoShelfFreq,
    kLoShelfGain,
    kHiShelfFreq,
    kHiShelfGain,
    kNotch1Freq,
    kNotch1Q,
    kNotch1Cut,
    kNotch2Freq,
    kNotch2Q,
    kNotch2Cut,
    kNotch3Freq,
    kNotch3Q,
    kNotch3Cut,
    kNotch4Freq,
    kNotch4Q,
    kNotch4Cut,
    kNotch5Freq,
    kNotch5Q,
    kNotch5Cut,
    kPreVolume,
    kBypass,
    kLoCutoffFreq,
    kLoCutoffQ,
    kHiCutoffFreq,
    kHiCutoffQ,
    kHDCDBypass,
    kHDCDGainScaleOff,
    kDDXCompBypass,
    kNotch6Freq,
    kNotch6Q,
    kNotch6Cut,
    kNotch7Freq,
    kNotch7Q,
    kNotch7Cut,
    kAnalogIn,
    kAnalogVolume,
    kDelay,
    kAllpassFreq,
    kAllpassQ,
    kBypassMask,
    kHiCutoff2Freq,
    kHiCutoff2Q,
    kNotch8Freq,
    kNotch8Q,
    kNotch8Cut,
    kNotch9Freq,
    kNotch9Q,
    kNotch9Cut,
    kNotchAFreq,
    kNotchAQ,
    kNotchACut,
    kDBNotchWidth
};

#define kUIArraySize  (kDBNotchWidth+1)

// orphaned control IDs
/*
#define kNotch6Freq   (10000+20)
#define kNotch6Q      (10000+21)
#define kNotch6Boost  (10000+22)
#define kNotch7Freq   (10000+23)
#define kNotch7Q      (10000+24)
#define kNotch7Boost  (10000+25)
#define kLoShelf2Freq (10000+26)
#define kLoShelf2Boost  (10000+27)
#define kHiShelf2Freq (10000+28)
#define kHiShelf2Boost  (10000+29)
```

```
*/

// DSP IDs
enum {
    kDSPMainVolume,
    kDSPLoShelfFreq,
    kDSPLoShelfGain,
    kDSPHiShelfFreq,
    kDSPHiShelfGain,
    kDSPNotch1Freq,
    kDSPNotch1Q,
    kDSPNotch1Cut,
    kDSPNotch2Freq,
    kDSPNotch2Q,
    kDSPNotch2Cut,
    kDSPNotch3Freq,
    kDSPNotch3Q,
    kDSPNotch3Cut,
    kDSPNotch4Freq,
    kDSPNotch4Q,
    kDSPNotch4Cut,
    kDSPNotch5Freq,
    kDSPNotch5Q,
    kDSPNotch5Cut,
    kDSPPreVolume,
    kDSPBypass,
    kDSPLoCutoffScale,
    kDSPLoCutoffFc,
    kDSPLoCutoffQc,
    kDSPHiCutoffScale,
    kDSPHiCutoffA2,
    kDSPHiCutoffA1,
    kDSPHDCDBypass,
    kDSPHDCDGainScaleOff,
    kDSPDDXCompBypass,
    kDSPNotch6Freq,
    kDSPNotch6Q,
    kDSPNotch6Cut,
    kDSPNotch7Freq,
    kDSPNotch7Q,
    kDSPNotch7Cut,
    kDSPAnalogIn,
    kDSPAnalogVol,
    kDSPBypassMask,
    kDSPDelay,
    kDSPAllpassFreq,
    kDSPAllpassQ,
    kDSP2HiCutoffScale,
    kDSP2HiCutoffA2,
    kDSP2HiCutoffA1,
    kDSPNotch8Freq,
    kDSPNotch8Q,
    kDSPNotch8Cut,
    kDSPNotch9Freq,
    kDSPNotch9Q,
    kDSPNotch9Cut,
    kDSPNotchAFreq,
    kDSPNotchAQ,
    kDSPNotchACut
};

#define kDSPArraySize (kDSPNotchACut+1)

// bit definitions for kDSPBypassMask above
enum {
  kBypassNotch1,
  kBypassNotch2,
  kBypassNotch3,
  kBypassNotch4,
  kBypassHipass,
  kBypassLopass,
  kBypassLoshelf,
```

```
    kBypassHiShelf,
    kBypassConecry,
    kBypassAllpass,
    kBypassDBNotch,
    kBypassNLopass
};
```

```
#if !defined(AFX_DBNOTCH_H__241CC3C5_14D9_11D3_96EE_006097CDB9E2__INCLUDED_)
#define AFX_DBNOTCH_H__241CC3C5_14D9_11D3_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// DBNotch.h : header file
//

class CTabDialog;

/////////////////////////////////////////////////////////////////////////////
// CDBNotch dialog

class CDBNotch : public CPropertyPage
{
  DECLARE_DYNCREATE(CDBNotch)

// Construction
public:
  CDBNotch();
  ~CDBNotch();

// Dialog Data
  //{{AFX_DATA(CDBNotch)
  enum { IDD = IDD_PP11 };
  CButton m_Bypass;
  CSliderCtrl m_CompGainSlider;
  CSliderCtrl m_CompQSlider;
  CSliderCtrl m_QSlider;
  CSliderCtrl m_FreqSlider;
  //}}AFX_DATA


// Overrides
  // ClassWizard generate virtual function overrides
  //{{AFX_VIRTUAL(CDBNotch)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  // Generated message map functions
  //{{AFX_MSG(CDBNotch)
  afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
  afx_msg void OnPaint();
  virtual BOOL OnInitDialog();
  afx_msg void OnBypass();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()

  CTabDialog *m_ParentWindow;
  void SendStringToUI(int which);

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DBNOTCH_H__241CC3C5_14D9_11D3_96EE_006097CDB9E2__INCLUDED_)
```

```
// DBNotch.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "DBNotch.h"
#include "TabDialog.h"
#include "DSP56kManager.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDBNotch property page

IMPLEMENT_DYNCREATE(CDBNotch, CPropertyPage)

CDBNotch::CDBNotch() : CPropertyPage(CDBNotch::IDD)
{
  //{{AFX_DATA_INIT(CDBNotch)
    // NOTE: the ClassWizard will add member initialization here
  //}}AFX_DATA_INIT
}

CDBNotch::~CDBNotch()
{
}

void CDBNotch::DoDataExchange(CDataExchange* pDX)
{
  CPropertyPage::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CDBNotch)
  DDX_Control(pDX, IDC_CHECK5, m_Bypass);
  DDX_Control(pDX, IDC_SLIDER6, m_CompGainSlider);
  DDX_Control(pDX, IDC_SLIDER5, m_CompQSlider);
  DDX_Control(pDX, IDC_SLIDER2, m_QSlider);
  DDX_Control(pDX, IDC_SLIDER1, m_FreqSlider);
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDBNotch, CPropertyPage)
  //{{AFX_MSG_MAP(CDBNotch)
  ON_WM_VSCROLL()
  ON_WM_PAINT()
  ON_BN_CLICKED(IDC_CHECK5, OnBypass)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDBNotch message handlers

void CDBNotch::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
  // TODO: Add your message handler code here and/or call default
  CSliderCtrl *slider = (CSliderCtrl *)pScrollBar;
  int which;

  CPropertyPage::OnVScroll(nSBCode, nPos, pScrollBar);

  Sleep(50);
  if( slider == &m_FreqSlider )
    which = kNotch9Freq;
  else if( slider == &m_QSlider )
    which = kNotch9Q;
  else if( slider == &m_CompQSlider )
    which = kNotch8Q;
  else if( slider == &m_CompGainSlider )
    which = kNotch8Cut;
```

```
  else
    return;
  g_DSPManager->SetParamValue(CONTROL_RANGE-slider->GetPos(),which);
  SendStringToUI(which);

}

void CDBNotch::OnPaint()
{
  CPaintDC dc(this); // device context for painting

  // TODO: Add your message handler code here
  m_FreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch9Freq));
  m_QSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch9Q));
  m_CompQSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch8Q));
  m_CompGainSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch8Cut));

  m_Bypass.SetCheck(g_DSPManager->GetBypassSection(kBypassDBNotch));

  // Do not call CPropertyPage::OnPaint() for painting messages
}

BOOL CDBNotch::OnInitDialog()
{
  CPropertyPage::OnInitDialog();

  // TODO: Add extra initialization here
  m_FreqSlider.SetRange(0,CONTROL_RANGE);
  m_FreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch9Freq));
  m_FreqSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_QSlider.SetRange(0,CONTROL_RANGE);
  m_QSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch9Q));
  m_QSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_CompQSlider.SetRange(0,CONTROL_RANGE);
  m_CompQSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch8Q));
  m_CompQSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_CompGainSlider.SetRange(0,CONTROL_RANGE);
  m_CompGainSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch8Cut));
  m_CompGainSlider.SetTicFreq((CONTROL_RANGE+1)/16);


  m_ParentWindow = (CTabDialog *) GetParent()->GetParent();

  return TRUE;  // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CDBNotch::SendStringToUI(int which)
{

    CString str;

    g_DSPManager->GetStringValue(which,str);
    m_ParentWindow->SetStatusString(0,str);

}


void CDBNotch::OnBypass()
{
  // TODO: Add your control notification handler code here
  int state = m_Bypass.GetState() & 0x3;

  g_DSPManager->SetBypassSection(state,kBypassDBNotch);

}
```

```
// MainPage.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "MainPage.h"

#include "TabDialog.h"
#include "DSP56kManager.h"
#include <stdlib.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainPage property page

IMPLEMENT_DYNCREATE(CMainPage, CPropertyPage)

CMainPage::CMainPage() : CPropertyPage(CMainPage::IDD)
{

  //{{AFX_DATA_INIT(CMainPage)
  m_BypassCheckBox = FALSE;
  m_HDCDBypass = FALSE;
  m_GainScaleBypass = FALSE;
  m_AnalogInput = FALSE;
  //}}AFX_DATA_INIT

}

CMainPage::~CMainPage()
{
}

void CMainPage::DoDataExchange(CDataExchange* pDX)
{
  CPropertyPage::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CMainPage)
  DDX_Control(pDX, IDC_SLIDER8, m_Delay);
  DDX_Control(pDX, IDC_SLIDER7, m_AnalogVol);
  DDX_Control(pDX, IDC_BUTTON1, m_ResetAll);
  DDX_Control(pDX, IDC_SLIDER3, m_PreVolumeSlider);
  DDX_Control(pDX, IDC_SLIDER1, m_VolumeSlider);
  DDX_Check(pDX, IDC_CHECK1, m_BypassCheckBox);
  DDX_Check(pDX, IDC_CHECK2, m_HDCDBypass);
  DDX_Check(pDX, IDC_CHECK3, m_GainScaleBypass);
  DDX_Check(pDX, IDC_CHECK4, m_AnalogInput);
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CMainPage, CPropertyPage)
  //{{AFX_MSG_MAP(CMainPage)
  ON_WM_VSCROLL()
  ON_WM_SHOWWINDOW()
  ON_BN_CLICKED(IDC_CHECK1, OnBypassButton)
  ON_WM_PAINT()
  ON_BN_CLICKED(IDC_CHECK2, OnHDCDBypass)
  ON_BN_CLICKED(IDC_CHECK3, OnGainScaleBypass)
  ON_BN_CLICKED(IDC_BUTTON1, OnResetAll)
  ON_BN_CLICKED(IDC_CHECK4, OnAnalogInput)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMainPage message handlers

void CMainPage::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
```

0051

```
{
  // TODO: Add your message handler code here and/or call default
  CSliderCtrl *slider = (CSliderCtrl *)pScrollBar;
  int which;

  CPropertyPage::OnVScroll(nSBCode, nPos, pScrollBar);

  Sleep(50);
  if ( slider == &m_VolumeSlider)
    which = kMainVolume;
  else if (slider == &m_PreVolumeSlider)
    which = kPreVolume;
  else if (slider == &m_AnalogVol)
    which = kAnalogVolume;
  else if (slider == &m_Delay)
    which = kDelay;
  else
    return;
  g_DSPManager->SetParamValue(CONTROL_RANGE-slider->GetPos(),which);
  SendStringToUI(which);

}

BOOL CMainPage::OnInitDialog()
{
  CPropertyPage::OnInitDialog();

  // TODO: Add extra initialization here
  m_VolumeSlider.SetRange(0,CONTROL_RANGE);
  m_VolumeSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kMainVolume));
  m_VolumeSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_VolumeSlider.SetPageSize(PG_CONTROL_AMT);

  m_PreVolumeSlider.SetRange(0,CONTROL_RANGE);
  m_PreVolumeSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kPreVolume));
  m_PreVolumeSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_PreVolumeSlider.SetPageSize(PG_CONTROL_AMT);

  m_AnalogVol.SetRange(0,CONTROL_RANGE);
  m_AnalogVol.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kAnalogVolume));
  m_AnalogVol.SetTicFreq((CONTROL_RANGE+1)/16);
  m_AnalogVol.SetPageSize(PG_CONTROL_AMT);

  m_Delay.SetRange(0,CONTROL_RANGE);
  m_Delay.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kAnalogVolume));
  m_Delay.SetTicFreq((CONTROL_RANGE+1)/16);
  m_Delay.SetPageSize(PG_CONTROL_AMT);

  m_BypassCheckBox = false;
  m_ParentWindow = (CTabDialog *) GetParent()->GetParent();

  return TRUE;  // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CMainPage::OnShowWindow(BOOL bShow, UINT nStatus)
{
  CPropertyPage::OnShowWindow(bShow, nStatus);

  // TODO: Add your message handler code here

}

void CMainPage::OnBypassButton()
{
  // TODO: Add your control notification handler code here
  m_BypassCheckBox = !m_BypassCheckBox;
  g_DSPManager->SetBypass(m_BypassCheckBox);

}

void CMainPage::OnPaint()
```

```
{
  CPaintDC dc(this); // device context for painting

  // TODO: Add your message handler code here
  m_VolumeSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kMainVolume));
  m_PreVolumeSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kPreVolume));
  m_AnalogVol.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kAnalogVolume));
  m_Delay.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kDelay));

  // Do not call CPropertyPage::OnPaint() for painting messages
}

void CMainPage::OnHDCDBypass()
{
  // TODO: Add your control notification handler code here
  m_HDCDBypass = !m_HDCDBypass;
  g_DSPManager->SetHDCDBypass(m_HDCDBypass);

}

void CMainPage::OnGainScaleBypass()
{
  // TODO: Add your control notification handler code here
  m_GainScaleBypass = !m_GainScaleBypass;
  g_DSPManager->SetHDCDGainScale(m_GainScaleBypass);

}

void CMainPage::OnResetAll()
{
  // TODO: Add your control notification handler code here

  // MM 8/3/99 Added extra dialog warning.
  if( MessageBox("Are you sure you want to reset all parameters?","WARNING",MB_YESNO) != IDYES )
    return;
  g_DSPManager->ResetAll();
  m_VolumeSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kMainVolume));
  m_PreVolumeSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kPreVolume));
  m_AnalogVol.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kAnalogVolume));
  m_Delay.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kDelay));

}

void CMainPage::OnAnalogInput()
{
  // TODO: Add your control notification handler code here
  m_AnalogInput = !m_AnalogInput;
  g_DSPManager->SetAnalogInput(m_AnalogInput);

}

void CMainPage::SendStringToUI(int which)
{

    CString str;

    g_DSPManager->GetStringValue(which,str);
    m_ParentWindow->SetStatusString(0,str);

}
```

```
#if !defined(AFX_MAINPAGE_H__90463DA4_D52E_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_MAINPAGE_H__90463DA4_D52E_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif

// _MSC_VER >= 1000
// MainPage.h : header file
//


class CTabDialog;

/////////////////////////////////////////////////////////////////////////////
// CMainPage dialog

class CMainPage : public CPropertyPage
{
  DECLARE_DYNCREATE(CMainPage)

// Construction
public:
  CMainPage();
  ~CMainPage();

// Dialog Data
  //{{AFX_DATA(CMainPage)
  enum { IDD = IDD_PP2 };
  CSliderCtrl m_Delay;
  CSliderCtrl m_AnalogVol;
  CButton m_ResetAll;
  CSliderCtrl m_PreVolumeSlider;
  CSliderCtrl m_VolumeSlider;
  BOOL  m_BypassCheckBox;
  BOOL  m_HDCDBypass;
  BOOL  m_GainScaleBypass;
  BOOL  m_AnalogInput;
  //}}AFX_DATA


// Overrides
  // ClassWizard generate virtual function overrides
  //{{AFX_VIRTUAL(CMainPage)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  // Generated message map functions
  //{{AFX_MSG(CMainPage)
  afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
  virtual BOOL OnInitDialog();
  afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
  afx_msg void OnBypassButton();
  afx_msg void OnPaint();
  afx_msg void OnHDCDBypass();
  afx_msg void OnGainScaleBypass();
  afx_msg void OnResetAll();
  afx_msg void OnAnalogInput();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()

  CTabDialog *m_ParentWindow;
  void SendStringToUI(int which);

private:

};

//{{AFX_INSERT_LOCATION}}
```

0054

```
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MAINPAGE_H__90463DA4_D52E_11D2_96EE_006097CDB9E2__INCLUDED_)
```

```
// I2CIPortComm.h: interface for the I2CIPortComm class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_I2CIPORTCOMM_H__33F9EF07_F6EF_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_I2CIPORTCOMM_H__33F9EF07_F6EF_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "DSPComm.h"


enum {
  MI2C_STATE_CLOSED,
  MI2C_STATE_PENDING_OPEN,
  MI2C_STATE_OPEN,
  MI2C_STATE_PENDING_TX,
  MI2C_STATE_PENDING_RX
};

#define MI2_BUFFER_SIZE (3*3000)      // should be multiple of 3

class I2CIPortComm : public DSPComm
{
  int   m_state;
  int   m_error_code;
  long  m_TxReq;
  char  m_tx_buffer[MI2_BUFFER_SIZE];
  long  m_tx_write,m_tx_read;
  long  m_detect;
  BOOL  m_get_detect;
  long  m_retryCount;
  BOOL  m_initialTry;
  HWND  m_HWnd;

public:
  virtual BOOL IsTransmitting(void);
  virtual long GetDetectState(void);
  virtual void MessageHandler(WPARAM iPortEventCode);
  virtual BOOL CheckState(void);
  I2CIPortComm(HWND p);
  virtual ~I2CIPortComm();
  virtual long SendDSPWord(long);
  virtual long SendDSPMemory(char *,long);


};

#endif // !defined(AFX_I2CIPORTCOMM_H__33F9EF07_F6EF_11D2_96EE_006097CDB9E2__INCLUDED_)
```

0056

```
// I2CIPortComm.cpp: implementation of the I2CIPortComm class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "sa.h"
#include "I2CIPortComm.h"
#include "COMPortChooser.h"

#include "i2c200.h"

static struct I2C_PROP i2c;
static I2CIPortComm *g_I2CComm = NULL;

#define I2C_SLAVE_ADDRESS 0xB0
#define I2C_MAX_AMOUNT  (5*3)     // don't send more than this in one-shot
#define I2C_RETRY_COUNT 0

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//static int cb_message;

void iPortMsgHandler(WPARAM iPortEventCode)
{

  if( g_I2CComm )
    g_I2CComm->MessageHandler(iPortEventCode);

}


//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////
/*static BOOL wait_for_message(void)
{

  long count;

  while( cb_message == -1 )
  {
    Sleep(100);
    count++;
    if( count > 20 )
      return( false );
  }
  return( true );

}*/

I2CIPortComm::I2CIPortComm(HWND p)
{

  int r;
  CCOMPortChooser dlg;

  dlg.DoModal();

  m_error_code = 0;
  m_state = MI2C_STATE_CLOSED;
  m_TxReq = 0;
  m_tx_write = m_tx_read = 0;
  m_detect = 0;
  m_get_detect = false;
  m_initialTry = false;
  m_HWnd = p;

  i2c.comport = dlg.which_port;
```

0057

```
  i2c.WmMsgNo = 0;
  i2c.pfCBF = iPortMsgHandler;
#ifdef  NDEBUG
  i2c.pcLogFileName = NULL;
#else
  i2c.pcLogFileName = "log.txt";
#endif
  i2c.LogFileLevel = 1;
  i2c.LogFileSize = 1000;
  i2c.HostSlaveAddr = 0x6E;
  i2c.BusTimeOut = 1000;
  i2c.MasterBitRate = 2;       // 100 kHz
  i2c.MasterRxBufSize = 512;
  i2c.MasterTxBufSize = MI2_BUFFER_SIZE;
  i2c.MasterArbRetry = 0;
  i2c.SlaveRxGCEnable = 1;
  i2c.SlaveRxBufSize = 512;
  i2c.SlaveTxBufSize = 512;

  r = I2COpen(m_HWnd,AfxGetInstanceHandle(),&i2c);
  if( r )
      m_error_code = 0x1000;
  m_state = MI2C_STATE_PENDING_OPEN;
  g_I2CComm = this;

}

I2CIPortComm::~I2CIPortComm()
{

  I2CClose();
  g_I2CComm = NULL;

}

long I2CIPortComm::SendDSPWord(long value)
{

  char  buffer[3];

  if( m_state < MI2C_STATE_OPEN )
    return( 0L );

  buffer[0] = (value >> 16) & 0xFF;
  buffer[1] = (value >> 8) & 0xFF;
  buffer[2] = value & 0xFF;
  return( SendDSPMemory(buffer,3) );

}

long I2CIPortComm::SendDSPMemory(char *data, long len)
{

  // make sure the link is valid
  if( m_state < MI2C_STATE_OPEN )
    return( 0L );

  long rval = 0L;
  long b;
  long tocopy,amount,freespace;

  // copy data into the tx buffer
  if( data )
  {
    freespace = m_tx_read - m_tx_write;
    if( freespace  <= 0 )
      freespace += MI2_BUFFER_SIZE;
    ASSERT( (len + 3) <= freespace );       // not enough space in the buffer
    for( tocopy = len ; tocopy > 0 ; tocopy -= amount )
    {
        if( m_tx_write >= m_tx_read )
        {
```

```
            amount = MI2_BUFFER_SIZE - m_tx_write;
          }
          else
          {
            amount = m_tx_read - m_tx_write;
          }
          amount = min(amount,tocopy);
          memcpy(m_tx_buffer+m_tx_write,data,amount);
          data += amount;
          m_tx_write += amount;
          m_tx_write %= MI2_BUFFER_SIZE;
        }
      }
      // attempt to send it
      if( m_state == MI2C_STATE_OPEN )
      {
        amount = m_tx_write - m_tx_read;
//      ASSERT( amount == ((amount / 3) * 3) );
        if( amount < 0 )
          amount = MI2_BUFFER_SIZE - m_tx_read;
//        amount = (amount / 3) * 3;      // Send complete DSP Words.
        amount = min(amount,I2C_MAX_AMOUNT);        // throttle
        b = I2CMasterTx(I2C_SLAVE_ADDRESS, (unsigned char *) m_tx_buffer+m_tx_read , amount, 1);
        if( b == 0 )
          {
            m_state = MI2C_STATE_PENDING_TX;
            m_TxReq = amount;
            if( !m_initialTry )
            {
              m_retryCount = 0;
              m_initialTry = true;
            }
          }
        else
        {
          if( m_error_code == 0 )
            m_error_code = b;
          ASSERT(b == 0);
        }
      }
      return( rval );

    }

    void I2CIPortComm::MessageHandler(WPARAM iPortEventCode)
    {

      struct I2C_PROP si2c;
      int r;
      long tx_count;
      long val;
      unsigned char buffer[10];

      switch( m_state )
        {
          case MI2C_STATE_PENDING_OPEN:
            if( iPortEventCode == I2C_OPEN_SUCCESSFUL )
            {
              m_state = MI2C_STATE_OPEN;
              if( m_tx_read != m_tx_write )
                SendDSPMemory(NULL,0L);
            }
            else
              m_error_code = iPortEventCode;

            break;

          case MI2C_STATE_PENDING_RX:
            if( iPortEventCode == I2C_MRX_COMPLETE )
            {
              r = I2CGetMasterRxMsg(3,buffer);
              if( r == 3 )
```

```
            {
              val = buffer[0] & 0xFF;
              val << 8;
              val = val | (buffer[1] & 0xFF);
              val << 8;
              val = val | (buffer[2] & 0xFF);
              m_detect = val;
            }
          }
          //else
          //  m_error_code = iPortEventCode;
          m_state = MI2C_STATE_OPEN;
          if( m_tx_read != m_tx_write )
            SendDSPMemory(NULL,0L);
          break;

        case MI2C_STATE_PENDING_TX:
          if( iPortEventCode == I2C_MTX_COMPLETE )
          {
            // MM 5/20/99 Don't call status here.
//            r = I2CGetStatus(&si2c);
//            tx_count = si2c.MasterTxByteCount;
            tx_count = m_TxReq;
            m_initialTry = false;
          }
          else
          {
            m_retryCount++;
            if( m_retryCount > I2C_RETRY_COUNT )
            {
              m_initialTry = false;
              if( m_error_code == 0 )
                m_error_code = iPortEventCode;
              //ASSERT(m_retryCount <= I2C_RETRY_COUNT);
              I2CClose();
              r = I2COpen(m_HWnd,AfxGetInstanceHandle(),&i2c);
              ASSERT( r == 0 );
              m_state = MI2C_STATE_PENDING_OPEN;
              return;
            }
            tx_count = 0;
          }
          m_state = MI2C_STATE_OPEN;
          m_tx_read += tx_count;
          m_tx_read %= MI2_BUFFER_SIZE;
          if( m_get_detect )
          {
            m_get_detect = false;
            r = I2CMasterRxExt(I2C_SLAVE_ADDRESS,3,1,1);
            if( r )
            {
              if( m_error_code == 0 )
                m_error_code = r;
            }
            else
              m_state = MI2C_STATE_PENDING_RX;
          }
          else if( m_tx_read != m_tx_write )
            SendDSPMemory(NULL,0L);
          break;

/*          case MI2C_STATE_PENDING_TX:
          if( iPortEventCode == I2C_MTX_COMPLETE )
          {
            r = I2CGetStatus(&si2c);
            if( m_TxReq == si2c.MasterTxByteCount )
            {
                m_state = MI2C_STATE_OPEN;
                m_tx_read += m_TxReq;
                m_tx_read %= MI2_BUFFER_SIZE;
                if( m_tx_read != m_tx_write )
                  SendDSPMemory(NULL,0L);
```

0060

```
        }
      }
      else
      {
        m_error_code = iPortEventCode;
        ;m_tx_read = m_tx_write = 0;
        m_state = MI2C_STATE_OPEN;
        if( m_tx_read != m_tx_write )
          SendDSPMemory(NULL,0L);
      }
      break; */
  }


}

BOOL I2CIPortComm::CheckState()
{

  return( m_state == MI2C_STATE_OPEN );

}

long I2CIPortComm::GetDetectState()
{

  long r,rval = 0;

  if( m_error_code )
  {
      rval = -m_error_code;
      m_error_code = 0;
  }
  else
  {
    rval = ( m_detect ) ? 1 : 0;
  }

  /*if( m_state == MI2C_STATE_PENDING_TX )
    m_get_detect = true;
  else if( m_state == MI2C_STATE_OPEN )
  {
    r = I2CMasterRxExt(I2C_SLAVE_ADDRESS,3,1,1);
    if( r )
      m_error_code = r;
    else
      m_state = MI2C_STATE_PENDING_RX;
  }*/
  return( rval );

}

BOOL I2CIPortComm::IsTransmitting()
{

  return( m_tx_write != m_tx_read );

}
```

0061

```
#if !defined(AFX_I2CDIALOG_H__B5D510E6_F7D5_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_I2CDIALOG_H__B5D510E6_F7D5_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// I2CDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CI2CDialog dialog

class CI2CDialog : public CDialog
{

    UINT            m_timerID;
    int             m_state;

// Construction
public:
  CI2CDialog(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
  //{{AFX_DATA(CI2CDialog)
  enum { IDD = IDD_DIALOG2 };
    // NOTE: the ClassWizard will add data members here
  //}}AFX_DATA


// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CI2CDialog)
  public:
  virtual BOOL OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO* pHandlerInfo);
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:

  // Generated message map functions
  //{{AFX_MSG(CI2CDialog)
  virtual BOOL OnInitDialog();
  afx_msg void OnTimer(UINT nIDEvent);
  afx_msg void OnClose();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_I2CDIALOG_H__B5D510E6_F7D5_11D2_96EE_006097CDB9E2__INCLUDED_)
```

0062

```
// I2CDialog.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "I2CDialog.h"
#include "DSP56kManager.h"

#define TIMERID 55

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CI2CDialog dialog


CI2CDialog::CI2CDialog(CWnd* pParent /*=NULL*/)
  : CDialog(CI2CDialog::IDD, pParent)
{
  //{{AFX_DATA_INIT(CI2CDialog)
    // NOTE: the ClassWizard will add member initialization here
  //}}AFX_DATA_INIT
}


void CI2CDialog::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CI2CDialog)
    // NOTE: the ClassWizard will add DDX and DDV calls here
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CI2CDialog, CDialog)
  //{{AFX_MSG_MAP(CI2CDialog)
  ON_WM_TIMER()
  ON_WM_CLOSE()
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CI2CDialog message handlers


BOOL CI2CDialog::OnInitDialog()
{
  CDialog::OnInitDialog();

  // TODO: Add extra initialization here
  g_DSPManager = new CDSP56kManager(this->m_hWnd);

  m_timerID = 0;
  m_timerID = SetTimer(TIMERID,50,NULL);
  m_state = 0;

  return TRUE;  // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CI2CDialog::OnTimer(UINT nIDEvent)
{
  // TODO: Add your message handler code here and/or call default
  if( nIDEvent == TIMERID )
    {
      if( g_DSPManager->IsReady() )
      {
          switch( m_state )
```

```
            {
            case 0:
              g_DSPManager->DownloadDSPCode();
              m_state++;
              break;
            case 1:
              g_DSPManager->ResetAll();
              EndDialog(IDOK);
              break;
            default:
              EndDialog(IDOK);
              break;
            }
        }
        else if( g_DSPManager->GetHDCDMode() < 0 )
          EndDialog(IDABORT);
      }
    else
      {
        CDialog::OnTimer(nIDEvent);
      }


}


BOOL CI2CDialog::OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO* pHandlerInfo)
{
  // TODO: Add your specialized code here and/or call the base class

  return CDialog::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}

void CI2CDialog::OnClose()
{
  // TODO: Add your message handler code here and/or call default
  if( m_timerID )
    KillTimer( m_timerID );

  CDialog::OnClose();
}
```

```
#if !defined(AFX_I2CCOMMERRORDIALOG_H__76C389E8_F889_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_I2CCOMMERRORDIALOG_H__76C389E8_F889_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// I2CCommErrorDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CI2CCommErrorDialog dialog

class CI2CCommErrorDialog : public CDialog
{
// Construction
public:
  CI2CCommErrorDialog(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
  //{{AFX_DATA(CI2CCommErrorDialog)
  enum { IDD = IDD_DIALOG4 };
  CString m_ErrorCode;
  //}}AFX_DATA


// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CI2CCommErrorDialog)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:

  // Generated message map functions
  //{{AFX_MSG(CI2CCommErrorDialog)
  afx_msg void OnButton1();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_I2CCOMMERRORDIALOG_H__76C389E8_F889_11D2_96EE_006097CDB9E2__INCLUDED_)
```

0065

```
// I2CCommErrorDialog.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "I2CCommErrorDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CI2CCommErrorDialog dialog


CI2CCommErrorDialog::CI2CCommErrorDialog(CWnd* pParent /*=NULL*/)
  : CDialog(CI2CCommErrorDialog::IDD, pParent)
{
  //{{AFX_DATA_INIT(CI2CCommErrorDialog)
  m_ErrorCode = _T("");
  //}}AFX_DATA_INIT
}


void CI2CCommErrorDialog::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CI2CCommErrorDialog)
  DDX_Text(pDX, IDC_EDIT1, m_ErrorCode);
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CI2CCommErrorDialog, CDialog)
  //{{AFX_MSG_MAP(CI2CCommErrorDialog)
  ON_BN_CLICKED(IDC_BUTTON1, OnButton1)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CI2CCommErrorDialog message handlers

void CI2CCommErrorDialog::OnButton1()
{
  // TODO: Add your control notification handler code here
  EndDialog(IDCANCEL);

}
```

0066

```
// PEQParam.h: interface for the CPEQParam class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_PEQPARAM_H__733FB7AB_4A49_11D3_96EE_006097CDB9E2__INCLUDED_)
#define AFX_PEQPARAM_H__733FB7AB_4A49_11D3_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

typedef struct _biquadcoef
{

  float c[6];   //-b0,b1,b2,a0,a1,a2

} biquadcoef;

class CPEQParam
{

  float   m_Gain;          // in dB (+ or -)
  float   m_Q;             // in Q units
  float   m_Frequency;     // in Hz
  float   m_SampleRate;    // in Hz
  double  m_pi;

public:
  void GetCoef(biquadcoef *);
  void GetAllpassCoef(biquadcoef *);
  void SetQ(float v) { m_Q = v; }
  void SetQ(CString &str);
  void SetGain(float v) { m_Gain = v; }
  void SetGain(CString &str);
  void SetFreq(float v) { m_Frequency = v; }
  void SetFreq(CString &str);

  CPEQParam();
  virtual ~CPEQParam();

};

#endif // !defined(AFX_PEQPARAM_H__733FB7AB_4A49_11D3_96EE_006097CDB9E2__INCLUDED_)
```

0067

```
// PEQParam.cpp: implementation of the CPEQParam class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "sa.h"
#include "PEQParam.h"
#include <stdio.h>
#include <math.h>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CPEQParam::CPEQParam()
  : m_Frequency(0.5),m_Gain(0.0),m_Q(1.0),m_SampleRate(44100.)
{
    m_pi = acos(-1.);
}

CPEQParam::~CPEQParam()
{

}

void CPEQParam::SetGain(CString & str)
{
    float v;

    sscanf(str,"%f",&v);
    m_Gain = v;
}

void CPEQParam::SetFreq(CString & str)
{

    float v;

    sscanf(str,"%f",&v);
    m_Frequency = v;
}

void CPEQParam::SetQ(CString & str)
{

    float v;

    sscanf(str,"%f",&v);
    m_Q = v;
}


void CPEQParam::GetAllpassCoef(biquadcoef *f)
{

  double gamma,beta,wc,tbeta;
  double normFreq = m_Frequency/m_SampleRate;

  wc = m_pi*normFreq;
  gamma = -cos(wc);
  tbeta = wc/(2*m_Q);
  if( tbeta > m_pi/4 )
    tbeta = m_pi/4;
  beta = (1.-tan(tbeta))/(1.+tan(tbeta));

  f->c[0] = (float) beta;             // b0
```

0068

```
  f->c[1] = (float) (gamma*(1+beta)); // b1
  f->c[2] = 1.;                       // b2
  f->c[3] = 1.;                       // a0
  f->c[4] = f->c[1];                  // a1
  f->c[5] = (float) beta;             // a2

}

void CPEQParam::GetCoef(biquadcoef *f)
{

  double M,L;
  biquadcoef af;
  double gain = (float) pow(10.,m_Gain/20.);

  GetAllpassCoef(&af);
  M = (1.-gain)/2.;
  L = (1.+gain)/2.;

  f->c[0] = (float) (af.c[0]*M+L);              // b0
  f->c[1] = (float) (af.c[1]*M+af.c[4]*L);  // b1
  f->c[2] = (float) (af.c[2]*M+af.c[5]*L);  // b2
  f->c[3] = af.c[3];
  f->c[4] = af.c[4];
  f->c[5] = af.c[5];

}
```

```
#if !defined(AFX_NOTCHPAGE2_H__5C8994C7_E2A4_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_NOTCHPAGE2_H__5C8994C7_E2A4_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// NotchPage2.h : header file
//

class CTabDialog;

/////////////////////////////////////////////////////////////////////////////
// CNotchPage2 dialog

class CNotchPage2 : public CPropertyPage
{
  DECLARE_DYNCREATE(CNotchPage2)

// Construction
public:
  CNotchPage2();
  ~CNotchPage2();

// Dialog Data
  //{{AFX_DATA(CNotchPage2)
  enum { IDD = IDD_PP9 };
  CButton m_BypassSecondButton;
  CButton m_BypassFirstButton;
  CSliderCtrl m_SliderCut4;
  CSliderCtrl m_SliderQ4;
  CSliderCtrl m_SliderFrequency4;
  CSliderCtrl m_SliderCut3;
  CSliderCtrl m_SliderQ3;
  CSliderCtrl m_SliderFrequency3;
  //}}AFX_DATA


// Overrides
  // ClassWizard generate virtual function overrides
  //{{AFX_VIRTUAL(CNotchPage2)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  // Generated message map functions
  //{{AFX_MSG(CNotchPage2)
  afx_msg void OnPaint();
  virtual BOOL OnInitDialog();
  afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
  afx_msg void OnBypassFirst();
  afx_msg void OnBypassSecond();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()

  CTabDialog *m_ParentWindow;
  void SendStringToUI(int which);

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_NOTCHPAGE2_H__5C8994C7_E2A4_11D2_96EE_006097CDB9E2__INCLUDED_)
```

```
// NotchPage2.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "NotchPage2.h"
#include "TabDialog.h"
#include "DSP56kManager.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CNotchPage2 property page

IMPLEMENT_DYNCREATE(CNotchPage2, CPropertyPage)

CNotchPage2::CNotchPage2() : CPropertyPage(CNotchPage2::IDD)
{
  //{{AFX_DATA_INIT(CNotchPage2)
  //}}AFX_DATA_INIT
}

CNotchPage2::~CNotchPage2()
{
}

void CNotchPage2::DoDataExchange(CDataExchange* pDX)
{
  CPropertyPage::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CNotchPage2)
  DDX_Control(pDX, IDC_CHECK6, m_BypassSecondButton);
  DDX_Control(pDX, IDC_CHECK5, m_BypassFirstButton);
  DDX_Control(pDX, IDC_SLIDER6, m_SliderCut4);
  DDX_Control(pDX, IDC_SLIDER5, m_SliderQ4);
  DDX_Control(pDX, IDC_SLIDER4, m_SliderFrequency4);
  DDX_Control(pDX, IDC_SLIDER3, m_SliderCut3);
  DDX_Control(pDX, IDC_SLIDER2, m_SliderQ3);
  DDX_Control(pDX, IDC_SLIDER1, m_SliderFrequency3);
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CNotchPage2, CPropertyPage)
  //{{AFX_MSG_MAP(CNotchPage2)
  ON_WM_PAINT()
  ON_WM_VSCROLL()
  ON_BN_CLICKED(IDC_CHECK5, OnBypassFirst)
  ON_BN_CLICKED(IDC_CHECK6, OnBypassSecond)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CNotchPage2 message handlers

void CNotchPage2::OnPaint()
{
  CPaintDC dc(this); // device context for painting

  // TODO: Add your message handler code here
  m_SliderFrequency3.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch6Freq));
  m_SliderQ3.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch6Q));
  m_SliderCut3.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch6Cut));
  m_SliderFrequency4.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch7Freq));
  m_SliderQ4.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch7Q));
  m_SliderCut4.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch7Cut));

  m_BypassFirstButton.SetCheck(g_DSPManager->GetBypassSection(kBypassNotch3));
  m_BypassSecondButton.SetCheck(g_DSPManager->GetBypassSection(kBypassNotch4));
```

0071

```
    // Do not call CPropertyPage::OnPaint() for painting messages
}

BOOL CNotchPage2::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    // TODO: Add extra initialization here
    m_SliderFrequency3.SetRange(0,CONTROL_RANGE);
    m_SliderFrequency3.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch6Freq));
    m_SliderFrequency3.SetTicFreq((CONTROL_RANGE+1)/16);
    m_SliderFrequency3.SetPageSize(PG_CONTROL_AMT);

    m_SliderQ3.SetRange(0,CONTROL_RANGE);
    m_SliderQ3.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch6Q));
    m_SliderQ3.SetTicFreq((CONTROL_RANGE+1)/16);
    m_SliderQ3.SetPageSize(PG_CONTROL_AMT);

    m_SliderCut3.SetRange(0,CONTROL_RANGE);
    m_SliderCut3.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch6Cut));
    m_SliderCut3.SetTicFreq((CONTROL_RANGE+1)/16);
    m_SliderCut3.SetPageSize(PG_CONTROL_AMT);

    m_SliderFrequency4.SetRange(0,CONTROL_RANGE);
    m_SliderFrequency4.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch7Freq));
    m_SliderFrequency4.SetTicFreq((CONTROL_RANGE+1)/16);
    m_SliderFrequency4.SetPageSize(PG_CONTROL_AMT);

    m_SliderQ4.SetRange(0,CONTROL_RANGE);
    m_SliderQ4.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch7Q));
    m_SliderQ4.SetTicFreq((CONTROL_RANGE+1)/16);
    m_SliderQ4.SetPageSize(PG_CONTROL_AMT);

    m_SliderCut4.SetRange(0,CONTROL_RANGE);
    m_SliderCut4.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch7Cut));
    m_SliderCut4.SetTicFreq((CONTROL_RANGE+1)/16);
    m_SliderCut4.SetPageSize(PG_CONTROL_AMT);

    m_ParentWindow = (CTabDialog *) GetParent()->GetParent();

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CNotchPage2::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    // TODO: Add your message handler code here and/or call default
    CSliderCtrl *slider = (CSliderCtrl *)pScrollBar;
    int which;

    CPropertyPage::OnVScroll(nSBCode, nPos, pScrollBar);

    Sleep(50);
    if( slider == &m_SliderFrequency3 )
      which = kNotch6Freq;
    else if( slider == &m_SliderQ3 )
      which = kNotch6Q;
    else if( slider == &m_SliderCut3 )
      which = kNotch6Cut;
    else if( slider == &m_SliderFrequency4 )
      which = kNotch7Freq;
    else if( slider == &m_SliderQ4 )
      which = kNotch7Q;
    else if( slider == &m_SliderCut4 )
      which = kNotch7Cut;
    else
      return;

    g_DSPManager->SetParamValue(CONTROL_RANGE-slider->GetPos(),which);
    SendStringToUI(which);
}
```

0072

```
void CNotchPage2::SendStringToUI(int which)
{

    CString str;

    g_DSPManager->GetStringValue(which,str);
    m_ParentWindow->SetStatusString(0,str);

}

void CNotchPage2::OnBypassFirst()
{
  // TODO: Add your control notification handler code here
  int state = m_BypassFirstButton.GetState() & 0x3;

  g_DSPManager->SetBypassSection(state,kBypassNotch3);

}

void CNotchPage2::OnBypassSecond()
{
  // TODO: Add your control notification handler code here
  int state = m_BypassSecondButton.GetState() & 0x3;

  g_DSPManager->SetBypassSection(state,kBypassNotch4);

}
```

```
#if !defined(AFX_NOTCHPAGE1_H__0CF7E208_D790_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_NOTCHPAGE1_H__0CF7E208_D790_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// NotchPage1.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CNotchPage1 dialog

class CTabDialog;

class CNotchPage1 : public CPropertyPage
{
  DECLARE_DYNCREATE(CNotchPage1)

// Construction
public:
  CNotchPage1();
  ~CNotchPage1();

// Dialog Data
  //{{AFX_DATA(CNotchPage1)
  enum { IDD = IDD_PP4 };
  CButton m_BypassSecondButton;
  CButton m_BypassFirstButton;
  CSliderCtrl m_SliderCut2;
  CSliderCtrl m_SliderQ2;
  CSliderCtrl m_SliderFrequency2;
  CSliderCtrl m_SliderCut1;
  CSliderCtrl m_SliderQ1;
  CSliderCtrl m_SliderFrequency1;
  //}}AFX_DATA


// Overrides
  // ClassWizard generate virtual function overrides
  //{{AFX_VIRTUAL(CNotchPage1)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  // Generated message map functions
  //{{AFX_MSG(CNotchPage1)
  virtual BOOL OnInitDialog();
  afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
  afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
  afx_msg void OnPaint();
  afx_msg void OnBypassFirst();
  afx_msg void OnBypassSecond();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()

  CTabDialog *m_ParentWindow;
  void SendStringToUI(int which);

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_NOTCHPAGE1_H__0CF7E208_D790_11D2_96EE_006097CDB9E2__INCLUDED_)
```

```
// NotchPage1.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "NotchPage1.h"
#include "TabDialog.h"
#include "DSP56kManager.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CNotchPage1 property page

IMPLEMENT_DYNCREATE(CNotchPage1, CPropertyPage)

CNotchPage1::CNotchPage1() : CPropertyPage(CNotchPage1::IDD)
{
  //{{AFX_DATA_INIT(CNotchPage1)
  //}}AFX_DATA_INIT
}

CNotchPage1::~CNotchPage1()
{
}

void CNotchPage1::DoDataExchange(CDataExchange* pDX)
{
  CPropertyPage::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CNotchPage1)
  DDX_Control(pDX, IDC_CHECK6, m_BypassSecondButton);
  DDX_Control(pDX, IDC_CHECK5, m_BypassFirstButton);
  DDX_Control(pDX, IDC_SLIDER6, m_SliderCut2);
  DDX_Control(pDX, IDC_SLIDER5, m_SliderQ2);
  DDX_Control(pDX, IDC_SLIDER4, m_SliderFrequency2);
  DDX_Control(pDX, IDC_SLIDER3, m_SliderCut1);
  DDX_Control(pDX, IDC_SLIDER2, m_SliderQ1);
  DDX_Control(pDX, IDC_SLIDER1, m_SliderFrequency1);
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CNotchPage1, CPropertyPage)
  //{{AFX_MSG_MAP(CNotchPage1)
  ON_WM_VSCROLL()
  ON_WM_SHOWWINDOW()
  ON_WM_PAINT()
  ON_BN_CLICKED(IDC_CHECK5, OnBypassFirst)
  ON_BN_CLICKED(IDC_CHECK6, OnBypassSecond)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CNotchPage1 message handlers

BOOL CNotchPage1::OnInitDialog()
{
  CPropertyPage::OnInitDialog();

  // TODO: Add extra initialization here
  m_SliderFrequency1.SetRange(0,CONTROL_RANGE);
  m_SliderFrequency1.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch1Freq));
  m_SliderFrequency1.SetTicFreq((CONTROL_RANGE+1)/16);
  m_SliderFrequency1.SetPageSize(PG_CONTROL_AMT);

  m_SliderQ1.SetRange(0,CONTROL_RANGE);
  m_SliderQ1.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch1Q));
  m_SliderQ1.SetTicFreq((CONTROL_RANGE+1)/16);
```

```
  m_SliderQ1.SetPageSize(PG_CONTROL_AMT);

  m_SliderCut1.SetRange(0,CONTROL_RANGE);
  m_SliderCut1.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch1Cut));
  m_SliderCut1.SetTicFreq((CONTROL_RANGE+1)/16);
  m_SliderCut1.SetPageSize(PG_CONTROL_AMT);

  m_SliderFrequency2.SetRange(0,CONTROL_RANGE);
  m_SliderFrequency2.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch2Freq));
  m_SliderFrequency2.SetTicFreq((CONTROL_RANGE+1)/16);
  m_SliderFrequency2.SetPageSize(PG_CONTROL_AMT);

  m_SliderQ2.SetRange(0,CONTROL_RANGE);
  m_SliderQ2.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch2Q));
  m_SliderQ2.SetTicFreq((CONTROL_RANGE+1)/16);
  m_SliderQ2.SetPageSize(PG_CONTROL_AMT);

  m_SliderCut2.SetRange(0,CONTROL_RANGE);
  m_SliderCut2.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch2Cut));
  m_SliderCut2.SetTicFreq((CONTROL_RANGE+1)/16);
  m_SliderCut2.SetPageSize(PG_CONTROL_AMT);

  m_ParentWindow = (CTabDialog *) GetParent()->GetParent();

  return TRUE;  // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CNotchPage1::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
  // TODO: Add your message handler code here and/or call default
  CSliderCtrl *slider = (CSliderCtrl *)pScrollBar;
  int which;

  CPropertyPage::OnVScroll(nSBCode, nPos, pScrollBar);

  Sleep(50);
  if( slider == &m_SliderFrequency1 )
    which = kNotch1Freq;
  else if( slider == &m_SliderQ1 )
    which = kNotch1Q;
  else if( slider == &m_SliderCut1 )
    which = kNotch1Cut;
  else if( slider == &m_SliderFrequency2 )
    which = kNotch2Freq;
  else if( slider == &m_SliderQ2 )
    which = kNotch2Q;
  else if( slider == &m_SliderCut2 )
    which = kNotch2Cut;
  else
    return;
  g_DSPManager->SetParamValue(CONTROL_RANGE-slider->GetPos(),which);
  SendStringToUI(which);

}

void CNotchPage1::OnShowWindow(BOOL bShow, UINT nStatus)
{
  CPropertyPage::OnShowWindow(bShow, nStatus);

  // TODO: Add your message handler code here

}

void CNotchPage1::OnPaint()
{
  CPaintDC dc(this); // device context for painting

  // TODO: Add your message handler code here
  m_SliderFrequency1.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch1Freq));
  m_SliderQ1.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch1Q));
  m_SliderCut1.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch1Cut));
```

```
    m_SliderFrequency2.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch2Freq));
    m_SliderQ2.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch2Q));
    m_SliderCut2.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch2Cut));
    m_BypassFirstButton.SetCheck(g_DSPManager->GetBypassSection(kBypassNotch1));
    m_BypassSecondButton.SetCheck(g_DSPManager->GetBypassSection(kBypassNotch2));

    // Do not call CPropertyPage::OnPaint() for painting messages
}

void CNotchPage1::SendStringToUI(int which)
{

    CString str;

    g_DSPManager->GetStringValue(which,str);
    m_ParentWindow->SetStatusString(0,str);

}

void CNotchPage1::OnBypassFirst()
{
    // TODO: Add your control notification handler code here
    int state = m_BypassFirstButton.GetState() & 0x3;

    g_DSPManager->SetBypassSection(state,kBypassNotch1);

}

void CNotchPage1::OnBypassSecond()
{
    // TODO: Add your control notification handler code here
    int state = m_BypassSecondButton.GetState() & 0x3;

    g_DSPManager->SetBypassSection(state,kBypassNotch2);

}
```

```
#if !defined(AFX_NEWCUTOFFPAGE_H__E2728226_E026_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_NEWCUTOFFPAGE_H__E2728226_E026_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// NewCutoffPage.h : header file
//

class CTabDialog;

/////////////////////////////////////////////////////////////////////////////
// CNewCutoffPage dialog

class CNewCutoffPage : public CPropertyPage
{
  DECLARE_DYNCREATE(CNewCutoffPage)

// Construction
public:
  CNewCutoffPage();
  ~CNewCutoffPage();

// Dialog Data
  //{{AFX_DATA(CNewCutoffPage)
  enum { IDD = IDD_PP8 };
  CButton m_BypassNLopassButton;
  CSliderCtrl m_NewHiQSlider;
  CSliderCtrl m_NewHiFreqSlider;
  CButton m_BypassLopassButton;
  CButton m_BypassHipassButton;
  CSliderCtrl m_HiQSlider;
  CSliderCtrl m_HiFreqSlider;
  CSliderCtrl m_LoQSlider;
  CSliderCtrl m_LoFreqSlider;
  //}}AFX_DATA


// Overrides
  // ClassWizard generate virtual function overrides
  //{{AFX_VIRTUAL(CNewCutoffPage)
  public:
  virtual BOOL OnSetActive();
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  // Generated message map functions
  //{{AFX_MSG(CNewCutoffPage)
  afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
  virtual BOOL OnInitDialog();
  afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
  afx_msg void OnPaint();
  afx_msg void OnBypassHipass();
  afx_msg void OnBypassLopass();
  afx_msg void OnBypassNewLopass();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()

  CTabDialog *m_ParentWindow;
  void SendStringToUI(int which);

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_NEWCUTOFFPAGE_H__E2728226_E026_11D2_96EE_006097CDB9E2__INCLUDED_)
```

0078

```
// NewCutoffPage.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "NewCutoffPage.h"
#include "TabDialog.h"
#include "DSP56kManager.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CNewCutoffPage property page

IMPLEMENT_DYNCREATE(CNewCutoffPage, CPropertyPage)

CNewCutoffPage::CNewCutoffPage() : CPropertyPage(CNewCutoffPage::IDD)
{
  //{{AFX_DATA_INIT(CNewCutoffPage)
  //}}AFX_DATA_INIT
}

CNewCutoffPage::~CNewCutoffPage()
{
}

void CNewCutoffPage::DoDataExchange(CDataExchange* pDX)
{
  CPropertyPage::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CNewCutoffPage)
  DDX_Control(pDX, IDC_CHECK7, m_BypassNLopassButton);
  DDX_Control(pDX, IDC_SLIDER9, m_NewHiQSlider);
  DDX_Control(pDX, IDC_SLIDER8, m_NewHiFreqSlider);
  DDX_Control(pDX, IDC_CHECK5, m_BypassLopassButton);
  DDX_Control(pDX, IDC_CHECK4, m_BypassHipassButton);
  DDX_Control(pDX, IDC_SLIDER5, m_HiQSlider);
  DDX_Control(pDX, IDC_SLIDER4, m_HiFreqSlider);
  DDX_Control(pDX, IDC_SLIDER2, m_LoQSlider);
  DDX_Control(pDX, IDC_SLIDER1, m_LoFreqSlider);
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CNewCutoffPage, CPropertyPage)
  //{{AFX_MSG_MAP(CNewCutoffPage)
  ON_WM_VSCROLL()
  ON_WM_SHOWWINDOW()
  ON_WM_PAINT()
  ON_BN_CLICKED(IDC_CHECK4, OnBypassHipass)
  ON_BN_CLICKED(IDC_CHECK5, OnBypassLopass)
  ON_BN_CLICKED(IDC_CHECK7, OnBypassNewLopass)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CNewCutoffPage message handlers

void CNewCutoffPage::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
  // TODO: Add your message handler code here and/or call default
  CSliderCtrl *slider = (CSliderCtrl *)pScrollBar;
  int which;

  CPropertyPage::OnVScroll(nSBCode, nPos, pScrollBar);

  Sleep(50);
  if( slider == &m_LoFreqSlider )
    which = kLoCutoffFreq;
```

0079

```
  else if( slider == &m_LoQSlider )
    which = kLoCutoffQ;
  else if( slider == &m_HiFreqSlider )
    which = kHiCutoffFreq;
  else if( slider == &m_HiQSlider )
    which = kHiCutoffQ;
  else if( slider == &m_NewHiFreqSlider )
    which = kHiCutoff2Freq;
  else if( slider == &m_NewHiQSlider )
    which = kHiCutoff2Q;
  else
    return;
  g_DSPManager->SetParamValue(CONTROL_RANGE-slider->GetPos(),which);
  SendStringToUI(which);

}

void CNewCutoffPage::SendStringToUI(int which)
{

    CString str;

    g_DSPManager->GetStringValue(which,str);
    m_ParentWindow->SetStatusString(0,str);

}

BOOL CNewCutoffPage::OnInitDialog()
{
  CPropertyPage::OnInitDialog();

  // TODO: Add extra initialization here
  m_LoFreqSlider.SetRange(0,CONTROL_RANGE);
  m_LoFreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kLoCutoffFreq));
  m_LoFreqSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_LoFreqSlider.SetPageSize(PG_CONTROL_AMT);

  m_LoQSlider.SetRange(0,CONTROL_RANGE);
  m_LoQSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kLoCutoffQ));
  m_LoQSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_LoQSlider.SetPageSize(PG_CONTROL_AMT);

  m_HiFreqSlider.SetRange(0,CONTROL_RANGE);
  m_HiFreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kHiCutoffFreq));
  m_HiFreqSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_HiFreqSlider.SetPageSize(PG_CONTROL_AMT);

  m_HiQSlider.SetRange(0,CONTROL_RANGE);
  m_HiQSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kHiCutoffQ));
  m_HiQSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_HiQSlider.SetPageSize(PG_CONTROL_AMT);

  m_NewHiFreqSlider.SetRange(0,CONTROL_RANGE);
  m_NewHiFreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kHiCutoff2Freq));
  m_NewHiFreqSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_NewHiFreqSlider.SetPageSize(PG_CONTROL_AMT);

  m_NewHiQSlider.SetRange(0,CONTROL_RANGE);
  m_NewHiQSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kHiCutoff2Q));
  m_NewHiQSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_NewHiQSlider.SetPageSize(PG_CONTROL_AMT);

  m_ParentWindow = (CTabDialog *) GetParent()->GetParent();

  return TRUE;  // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CNewCutoffPage::OnShowWindow(BOOL bShow, UINT nStatus)
{
  CPropertyPage::OnShowWindow(bShow, nStatus);
```

0080

```
  // TODO: Add your message handler code here

}

BOOL CNewCutoffPage::OnSetActive()
{
  // TODO: Add your specialized code here and/or call the base class

  return CPropertyPage::OnSetActive();
}

void CNewCutoffPage::OnPaint()
{
  CPaintDC dc(this); // device context for painting

  // TODO: Add your message handler code here
  m_LoFreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kLoCutoffFreq));
  m_LoQSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kLoCutoffQ));
  m_HiFreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kHiCutoffFreq));
  m_HiQSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kHiCutoffQ));
  m_NewHiFreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kHiCutoff2Freq));
  m_NewHiQSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kHiCutoff2Q));

  m_BypassHipassButton.SetCheck(g_DSPManager->GetBypassSection(kBypassHipass));
  m_BypassLopassButton.SetCheck(g_DSPManager->GetBypassSection(kBypassLopass));
  m_BypassNLopassButton.SetCheck(g_DSPManager->GetBypassSection(kBypassNLopass));
  // Do not call CPropertyPage::OnPaint() for painting messages
}

void CNewCutoffPage::OnBypassHipass()
{
  // TODO: Add your control notification handler code here
  int state = m_BypassHipassButton.GetState() & 0x3;

  g_DSPManager->SetBypassSection(state,kBypassHipass);

}

void CNewCutoffPage::OnBypassLopass()
{
  // TODO: Add your control notification handler code here
  int state = m_BypassLopassButton.GetState() & 0x3;

  g_DSPManager->SetBypassSection(state,kBypassLopass);

}

void CNewCutoffPage::OnBypassNewLopass()
{
  // TODO: Add your control notification handler code here
  int state = m_BypassNLopassButton.GetState() & 0x3;

  g_DSPManager->SetBypassSection(state,kBypassNLopass);

}
```

0081

```
// ShelfEQParam.h: interface for the CShelfEQParam class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_ShelfEQPARAM_H__733FB7AB_4A49_11D3_96EE_006097CDB9E2__INCLUDED_)
#define AFX_ShelfEQPARAM_H__733FB7AB_4A49_11D3_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

typedef struct _biquadcoef
{

  float c[6]; ` // b0,b1,b2,a0,a1,a2

} biquadcoef;

class CShelfEQParam
{

  float   m_Gain;          // in dB (+ or -)
  float   m_Frequency;     // in Hz
  float   m_SampleRate;    // in Hz
  double  m_pi;
  bool    m_HiShelf;

public:
  void GetCoef(biquadcoef *);
  void GetAllpassCoef(biquadcoef *);
  void SetGain(float v) { m_Gain = v; }
  void SetGain(CString &str);
  void SetFreq(float v) { m_Frequency = v; }
  void SetFreq(CString &str);

  CShelfEQParam(bool hi);
  virtual ~CShelfEQParam();

};

#endif // !defined(AFX_ShelfEQPARAM_H__733FB7AB_4A49_11D3_96EE_006097CDB9E2__INCLUDED_)
```

0082

```
// ShelfEQParam.cpp: implementation of the CShelfEQParam class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "sa.h"
#include "ShelfEQParam.h"
#include <stdio.h>
#include <math.h>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CShelfEQParam::CShelfEQParam(bool hi)
  : m_Frequency(0.5),m_Gain(0.0),m_SampleRate(44100.),m_HiShelf(hi)
{
    m_pi = acos(-1.);
}

CShelfEQParam::~CShelfEQParam()
{

}

void CShelfEQParam::SetGain(CString & str)
{
    float v;

    sscanf(str,"%f",&v);
    m_Gain = v;
}

void CShelfEQParam::SetFreq(CString & str)
{

    float v;

    sscanf(str,"%f",&v);
    m_Frequency = v;
}


void CShelfEQParam::GetAllpassCoef(biquadcoef *f)
{

  double gamma,wc;
  double normFreq = m_Frequency/m_SampleRate/2;

  wc = m_pi*normFreq;
  gamma = (tan(wc/2)-1)/(tan(wc/2)+1);

  if( m_HiShelf )
  {
    f->c[0] = (float) gamma;      // b0
    f->c[1] = 1.0;                // b1
  }
  else
  {
    f->c[0] = (float) -gamma;     // b0
    f->c[1] = -1.0;               // b1
  }

  f->c[2] = 0;                        // b2
  f->c[3] = 1.;                       // a0
  f->c[4] = (float) gamma;            // a1
```

```
  f->c[5] = 0;                          // a2

}

void CShelfEQParam::GetCoef(biquadcoef *f)
{

  double M,L;
  biquadcoef af;
  double gain = (float) pow(10.,m_Gain/20.);

  GetAllpassCoef(&af);
  M = (1.-gain)/2.;
  L = (1.+gain)/2.;

  f->c[0] = (float) (af.c[0]*M+L);              // b0
  f->c[1] = (float) (af.c[1]*M+af.c[4]*L);  // b1
  f->c[2] = (float) (af.c[2]*M+af.c[5]*L);  // b2
  f->c[3] = af.c[3];
  f->c[4] = af.c[4];
  f->c[5] = af.c[5];

}
```

```
// saDlg.h : header file
//

#if !defined(AFX_SADLG_H__33D93B0B_D0B8_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_SADLG_H__33D93B0B_D0B8_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CSaDlg dialog

class CSaDlg : public CDialog
{
// Construction
public:
  CSaDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
  //{{AFX_DATA(CSaDlg)
  enum { IDD = IDD_SA_DIALOG };
    // NOTE: the ClassWizard will add data members here
  //}}AFX_DATA

  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CSaDlg)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  HICON m_hIcon;

  // Generated message map functions
  //{{AFX_MSG(CSaDlg)
  virtual BOOL OnInitDialog();
  afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
  afx_msg void OnPaint();
  afx_msg HCURSOR OnQueryDragIcon();
  afx_msg void OnButton1();
  afx_msg void OnButton2();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SADLG_H__33D93B0B_D0B8_11D2_96EE_006097CDB9E2__INCLUDED_)
```

0085

```
// saDlg.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "saDlg.h"

#include "unit_ppi.h"
#include "functs.h"
#include "TabDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


static    int cmd_delay = 1000;
static    int io_delay = 50;


/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
  CAboutDlg();

// Dialog Data
  //{{AFX_DATA(CAboutDlg)
  enum { IDD = IDD_ABOUTBOX };
  //}}AFX_DATA

  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CAboutDlg)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  //{{AFX_MSG(CAboutDlg)
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
  //{{AFX_DATA_INIT(CAboutDlg)
  //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CAboutDlg)
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
  //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSaDlg dialog

CSaDlg::CSaDlg(CWnd* pParent /*=NULL*/)
  : CDialog(CSaDlg::IDD, pParent)
```

0086

```
{
  //{{AFX_DATA_INIT(CSaDlg)
    // NOTE: the ClassWizard will add member initialization here
  //}}AFX_DATA_INIT
  // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
  m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CSaDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CSaDlg)
    // NOTE: the ClassWizard will add DDX and DDV calls here
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CSaDlg, CDialog)
  //{{AFX_MSG_MAP(CSaDlg)
  ON_WM_SYSCOMMAND()
  ON_WM_PAINT()
  ON_WM_QUERYDRAGICON()
  ON_BN_CLICKED(IDC_BUTTON1, OnButton1)
  ON_BN_CLICKED(IDC_BUTTON2, OnButton2)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSaDlg message handlers

BOOL CSaDlg::OnInitDialog()
{
  CDialog::OnInitDialog();

  // Add "About..." menu item to system menu.

  // IDM_ABOUTBOX must be in the system command range.
  ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
  ASSERT(IDM_ABOUTBOX < 0xF000);

  CMenu* pSysMenu = GetSystemMenu(FALSE);
  if (pSysMenu != NULL)
  {
    CString strAboutMenu;
    strAboutMenu.LoadString(IDS_ABOUTBOX);
    if (!strAboutMenu.IsEmpty())
    {
      pSysMenu->AppendMenu(MF_SEPARATOR);
      pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
    }
  }

  // Set the icon for this dialog.  The framework does this automatically
  //  when the application's main window is not a dialog
  SetIcon(m_hIcon, TRUE);     // Set big icon
  SetIcon(m_hIcon, FALSE);    // Set small icon

  // TODO: Add extra initialization here
  if( load_dll()==false )
  {
    MessageBox("UNIT_PPI not found in the system directory. Fix this and restart application.");
    return false;
  }
  init_port_spi(1);
  set_cmd_delay_cnt_value(cmd_delay);
  set_io_delay_cnt_value(io_delay);

  return TRUE;  // return TRUE  unless you set the focus to a control
}

void CSaDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
  if ((nID & 0xFFF0) == IDM_ABOUTBOX)
```

0087

```
  {
    CAboutDlg dlgAbout;
    dlgAbout.DoModal();
  }
  else
  {
    CDialog::OnSysCommand(nID, lParam);
  }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CSaDlg::OnPaint()
{
  if (IsIconic())
  {
    CPaintDC dc(this); // device context for painting

    SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

    // Center icon in client rectangle
    int cxIcon = GetSystemMetrics(SM_CXICON);
    int cyIcon = GetSystemMetrics(SM_CYICON);
    CRect rect;
    GetClientRect(&rect);
    int x = (rect.Width() - cxIcon + 1) / 2;
    int y = (rect.Height() - cyIcon + 1) / 2;

    // Draw the icon
    dc.DrawIcon(x, y, m_hIcon);
  }
  else
  {
    CDialog::OnPaint();
  }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CSaDlg::OnQueryDragIcon()
{
  return (HCURSOR) m_hIcon;
}

void CSaDlg::OnButton1()
{

  long tlong;

  tlong=spi_xchng24(0xAA55AA);
  tlong=spi_xchng24(0x0);

}

void CSaDlg::OnButton2()
{
//  long tlong;
//
//  tlong=spi_xchng24(0x654321);
//  tlong=spi_xchng24(0x0);

    CTabDialog dlg;

    dlg.DoModal();

}
```

0088

```
// sa.h : main header file for the SA application
//

#if !defined(AFX_SA_H__33D93B09_D0B8_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_SA_H__33D93B09_D0B8_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
  #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"   // main symbols

/////////////////////////////////////////////////////////////////////////////
// CSaApp:
// See sa.cpp for the implementation of this class
//

class CSaApp : public CWinApp
{
public:
  CSaApp();

// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CSaApp)
  public:
  virtual BOOL InitInstance();
  //}}AFX_VIRTUAL

// Implementation

  //{{AFX_MSG(CSaApp)
    // NOTE - the ClassWizard will add and remove member functions here.
    //    DO NOT EDIT what you see in these blocks of generated code !
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SA_H__33D93B09_D0B8_11D2_96EE_006097CDB9E2__INCLUDED_)
```

0089

```
// sa.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "sa.h"
#include "saDlg.h"
#include "TabDialog.h"
#include "I2CDialog.h"
#include "DSP56kManager.h"

CDSP56kManager *g_DSPManager;

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSaApp

BEGIN_MESSAGE_MAP(CSaApp, CWinApp)
  //{{AFX_MSG_MAP(CSaApp)
    // NOTE - the ClassWizard will add and remove mapping macros here.
    //    DO NOT EDIT what you see in these blocks of generated code!
  //}}AFX_MSG
  ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSaApp construction

CSaApp::CSaApp()
{
  // TODO: add construction code here,
  // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CSaApp object

CSaApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CSaApp initialization

BOOL CSaApp::InitInstance()
{

  AfxEnableControlContainer();

  // Standard initialization
  // If you are not using these features and wish to reduce the size
  //  of your final executable, you should remove from the following
  //  the specific initialization routines you do not need.

#ifdef _AFXDLL
  Enable3dControls();     // Call this when using MFC in a shared DLL
#else
  Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif

  // Set dialog background color to black
//  SetDialogBkColor(RGB(0,0,0),RGB(255,255,255));

  int response;
  CI2CDialog  idlg;
  try
  {

    if( (response = idlg.DoModal()) != IDOK )
      {
```

0030

```
        if( response == IDABORT )
        {
          AfxMessageBox("Trouble initializing");
          Sleep(1000);
        }
//        return FALSE;
      }

  }
  catch(CException* e )
  {
//        printf("Trouble initializing\n");
//        Sleep(1000);
        return FALSE;
  }


  //  CSaDlg dlg;
  CTabDialog  dlg;

  m_pMainWnd = &dlg;
  int nResponse = dlg.DoModal();
  if (nResponse == IDOK)
  {
    // TODO: Place code here to handle when the dialog is
    //  dismissed with OK
  }
  else if (nResponse == IDCANCEL)
  {
    // TODO: Place code here to handle when the dialog is
    //  dismissed with Cancel
  }

  // Since the dialog has been closed, return FALSE so that we exit the
  //  application, rather than start the application's message pump.
  return FALSE;
}
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by sa.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_SA_DIALOG                   102
#define IDR_MAINFRAME                   128
#define IDD_DIALOG1                     129
#define IDD_PP1                         130
#define IDR_MENU1                       130
#define IDD_PP2                         131
#define IDD_PP3                         132
#define IDD_PP4                         133
#define IDD_PP5                         134
#define IDD_PP6                         135
#define IDD_PP7                         136
#define IDD_PP8                         137
#define IDD_PP9                         138
#define IDD_DIALOG2                     139
#define IDD_DIALOG3                     140
#define IDD_DIALOG4                     141
#define IDD_PP10                        142
#define IDD_PP11                        143
#define IDC_BUTTON1                     1001
#define IDC_BUTTON2                     1002
#define IDC_SLIDER1                     1004
#define IDC_CHECK1                      1005
#define IDC_SLIDER2                     1005
#define IDC_SLIDER3                     1006
#define IDC_RADIO1                      1006
#define IDC_SLIDER4                     1007
#define IDC_CHECK2                      1007
#define IDC_RADIO2                      1007
#define IDC_EDIT1                       1007
#define IDC_SLIDER5                     1008
#define IDC_CHECK3                      1008
#define IDC_RADIO3                      1008
#define IDC_SLIDER6                     1009
#define IDC_CHECK4                      1009
#define IDC_RADIO4                      1009
#define IDC_SLIDER7                     1010
#define IDC_CHECK5                      1010
#define IDC_SLIDER8                     1011
#define IDC_CHECK6                      1011
#define IDC_SLIDER9                     1012
#define IDC_CHECK7                      1013
#define ID_MENUITEM32771                32771
#define ID_MENUITEM32772                32772
#define ID_FILE_EXPORTPARAMFILE         32773

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        142
#define _APS_NEXT_COMMAND_VALUE         32774
#define _APS_NEXT_CONTROL_VALUE         1011
#define _APS_NEXT_SYMED_VALUE           103
#endif
#endif
```

0092

```
#if !defined(AFX_TABDIALOG_H__5CB3DF04_D20F_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_TABDIALOG_H__5CB3DF04_D20F_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// TabDialog.h : header file
//

#include "Page1.h"
#include "MainPage.h"
#include "DDX.h"
#include "NotchPage1.h"
#include "NotchPage2.h"
#include "StWaveRejPage.h"
#include "ShelvPage.h"
#include "NewCutoffPage.h"
#include "AllpassPage.h"
#include "DBNotch.h"

/////////////////////////////////////////////////////////////////////////////
// CTabDialog dialog


class CTabDialog : public CDialog
{
// Construction
public:
  void DisplayI2CState(long error);
  virtual void SetStatusString(int which,CString &s);
  CTabDialog(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
  //{{AFX_DATA(CTabDialog)
  enum { IDD = IDD_DIALOG1 };
    // NOTE: the ClassWizard will add data members here
  //}}AFX_DATA


// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CTabDialog)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  virtual void DisplayMode(int);

  CStatusBar      m_StatusBar;
  CPropertySheet  m_dlgPropSheet;
  HICON           m_hIcon;
//  CPage1          m_Page1;
  CMainPage       m_MainPage;
  CDDX            m_DDXPage;
  CNotchPage1     m_Notch1Page;
  CNotchPage2     m_Notch2Page;
  CStWaveRejPage  m_StWaveRejPage;
  CShelvPage      m_ShelvPage;
//  CCutoffPage     m_CutoffPage;
  CNewCutoffPage  m_CutoffPage;
  CAllpassPage    m_AllpassPage;
  CDBNotch        m_DBNotch;
  CBrush          m_brush;
  UINT            m_timerID;
  BOOL            m_HDCDDisplay;
  int             m_AIdx,m_ICnt;
  int             m_ErrorCounter;

  // Generated message map functions
  //{{AFX_MSG(CTabDialog)
```

0093

```
  virtual BOOL OnInitDialog();
  afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
  afx_msg void OnPaint();
  afx_msg HCURSOR OnQueryDragIcon();
  afx_msg void OnClose();
  afx_msg void OnMove(int x, int y);
  afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor);
  afx_msg void OnOpenMenu();
  afx_msg void OnSaveMenu();
  afx_msg void OnTimer(UINT nIDEvent);
  afx_msg void OnFileExportparamfile();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};


//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_TABDIALOG_H__5CB3DF04_D20F_11D2_96EE_006097CDB9E2__INCLUDED_)
```

```
// TabDialog.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "TabDialog.h"
#include "DSP56kManager.h"
#include "I2CCommErrorDialog.h"
#include <afxdlgs.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


#define TIMERID 1
#define ID_HDCD_DETECT    56000
#define ID_ERROR_DISPLAY  (ID_HDCD_DETECT+1)

static UINT auIDStatusBar[] = {
   ID_SEPARATOR,
   ID_HDCD_DETECT,
   ID_SEPARATOR,
   ID_ERROR_DISPLAY
};


#define AARRAYSIZE  16
static  char  *Detect_Array[AARRAYSIZE] = {
"            HDCD",
"           HDCD ",
"          HDCD  ",
"         HDCD   ",
"        HDCD    ",
"       HDCD     ",
"      HDCD      ",
"     HDCD       ",
"    HDCD        ",
"   HDCD         ",
"  HDCD          ",
" HDCD           ",
"HDCD            ",
"DCD            H",
"CD            HD",
"D            HDC"
};


/////////////////////////////////////////////////////////////////////////////
// CTabAboutDlg dialog used for App About

class CTabAboutDlg : public CDialog
{
public:
  CTabAboutDlg();

// Dialog Data
  //{{AFX_DATA(CTabAboutDlg)
  enum { IDD = IDD_ABOUTBOX };
  //}}AFX_DATA

  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CTabAboutDlg)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  //{{AFX_MSG(CTabAboutDlg)
```

0095

```
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

CTabAboutDlg::CTabAboutDlg() : CDialog(CTabAboutDlg::IDD)
{
  //{{AFX_DATA_INIT(CTabAboutDlg)
  //}}AFX_DATA_INIT
}

void CTabAboutDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CTabAboutDlg)
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CTabAboutDlg, CDialog)
  //{{AFX_MSG_MAP(CTabAboutDlg)
    // No message handlers
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTabDialog dialog


CTabDialog::CTabDialog(CWnd* pParent /*=NULL*/)
  : CDialog(CTabDialog::IDD, pParent)
{
  //{{AFX_DATA_INIT(CTabDialog)
    // NOTE: the ClassWizard will add member initialization here
  //}}AFX_DATA_INIT
  m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}


void CTabDialog::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CTabDialog)
    // NOTE: the ClassWizard will add DDX and DDV calls here
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CTabDialog, CDialog)
  //{{AFX_MSG_MAP(CTabDialog)
  ON_WM_SYSCOMMAND()
  ON_WM_PAINT()
  ON_WM_QUERYDRAGICON()
  ON_WM_CLOSE()
  ON_WM_MOVE()
  ON_WM_CTLCOLOR()
  ON_COMMAND(ID_MENUITEM32771, OnOpenMenu)
  ON_COMMAND(ID_MENUITEM32772, OnSaveMenu)
  ON_WM_TIMER()
  ON_COMMAND(ID_FILE_EXPORTPARAMFILE, OnFileExportparamfile)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTabDialog message handlers

#define PS_WIDTH  222
#define PS_HEIGHT 215

BOOL CTabDialog::OnInitDialog()
{

  int partsList[] = { 50, 100,  150,  -1};
```

0096

```
CDialog::OnInitDialog();


// Add "About..." menu item to system menu.

// IDM_ABOUTBOX must be in the system command range.
ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
ASSERT(IDM_ABOUTBOX < 0xF000);

CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
  CString strAboutMenu;
  strAboutMenu.LoadString(IDS_ABOUTBOX);
  if (!strAboutMenu.IsEmpty())
  {
    pSysMenu->AppendMenu(MF_SEPARATOR);
    pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
  }
}

// Set the icon for this dialog.  The framework does this automatically
//  when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);     // Set big icon
SetIcon(m_hIcon, FALSE);    // Set small icon

// build brush
m_brush.CreateSolidBrush(RGB(0,0,0));

// build prop sheet
m_dlgPropSheet.AddPage(&m_MainPage);
m_dlgPropSheet.AddPage(&m_DDXPage);
m_dlgPropSheet.AddPage(&m_ShelvPage);
m_dlgPropSheet.AddPage(&m_CutoffPage);
m_dlgPropSheet.AddPage(&m_Notch1Page);
m_dlgPropSheet.AddPage(&m_Notch2Page);
m_dlgPropSheet.AddPage(&m_StWaveRejPage);
m_dlgPropSheet.AddPage(&m_AllpassPage);
m_dlgPropSheet.AddPage(&m_DBNotch);

m_dlgPropSheet.Create(this, WS_CHILD | WS_VISIBLE, 0);
m_dlgPropSheet.ModifyStyleEx(0, WS_EX_CONTROLPARENT);
m_dlgPropSheet.ModifyStyle(0, WS_TABSTOP);

m_dlgPropSheet.SetWindowPos(NULL, 0, 0, PS_WIDTH, PS_HEIGHT, SWP_NOZORDER | SWP_NOSIZE | SWP_NOACTIVATE );

  // resize dialog to fit propsheet exactly
RECT rect;
m_dlgPropSheet.GetWindowRect(&rect);
//rect.bottom += 14;
//rect.right += 14;
SetWindowPos(&wndBottom,rect.left,rect.top,rect.right,rect.bottom, SWP_NOZORDER | SWP_NOMOVE );

// Add status bar
RECT sizerect;
sizerect.top = PS_HEIGHT;
sizerect.bottom = sizerect.top + 20;
sizerect.left = 0;
sizerect.right = PS_WIDTH;
//m_StatusBar.Create(WS_CHILD | WS_VISIBLE | CCS_BOTTOM, sizerect , this, AFX_IDW_STATUS_BAR );
//m_StatusBar.ShowWindow(SW_SHOWNA);
//m_StatusBar.SetSimple();
//m_StatusBar.SetText("Hello world!",255,0);
m_StatusBar.Create(this);
//m_StatusBar.SetIndicators(auIDStatusBar, sizeof(auIDStatusBar)/sizeof(UINT));
//m_StatusBar.SetPaneInfo(0, m_StatusBar.GetItemID(0), SBPS_STRETCH, NULL );
//m_StatusBar.GetStatusBarCtrl().SetText("Hello world!",0,0);

CRect rcClientStart;
CRect rcClientNow;
GetClientRect(rcClientStart);
```

0097

```
  RepositionBars(AFX_IDW_CONTROLBAR_FIRST, AFX_IDW_CONTROLBAR_LAST,
          0, reposQuery, rcClientNow);

  // Now move all the controls so they are in the same relative
  // position within the remaining client area as they would be
  // with no control bars.
/*  CPoint ptOffset(rcClientNow.left - rcClientStart.left,
          rcClientNow.top - rcClientStart.top);

  CRect  rcChild;
  CWnd* pwndChild = GetWindow(GW_CHILD);
  while (pwndChild)
  {
    pwndChild->GetWindowRect(rcChild);
    ScreenToClient(rcChild);
    rcChild.OffsetRect(ptOffset);
    pwndChild->MoveWindow(rcChild, FALSE);
    pwndChild = pwndChild->GetNextWindow();
  } */

  // Adjust the dialog window dimensions
  CRect rcWindow;
  GetWindowRect(rcWindow);
  rcWindow.right += rcClientStart.Width() - rcClientNow.Width();
  rcWindow.bottom += rcClientStart.Height() - rcClientNow.Height();
  rcWindow.bottom += 5;
  MoveWindow(rcWindow, FALSE);

  // And position the control bars
  RepositionBars(AFX_IDW_CONTROLBAR_FIRST, AFX_IDW_CONTROLBAR_LAST, 0);

  m_timerID = 0;
  m_timerID = SetTimer(TIMERID,50,NULL);
  m_ErrorCounter = 0;

  m_HDCDDisplay = false;
  DisplayMode(1);
  SetWindowText("Untitled");
  partsList[0] = rcClientNow.Width()/4;
  partsList[1] = partsList[0] + (rcClientNow.Width()/4);
  partsList[2] = partsList[1] + (rcClientNow.Width()/4);
  partsList[3] = -1;
  m_StatusBar.GetStatusBarCtrl().SetParts(sizeof(partsList)/sizeof(int),partsList);


  return TRUE;  // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CTabDialog::OnClose()
{

//  if( m_Page1 )
//    delete  m_Page1;
//  if( m_Page2 )
//    delete  m_Page2;
//  if( m_Page3 )
//    delete  m_Page3;

  if( m_timerID )
    KillTimer( m_timerID );

  delete g_DSPManager;

  CDialog::OnClose();
}

void CTabDialog::OnSysCommand(UINT nID, LPARAM lParam)
{
  if ((nID & 0xFFF0) == IDM_ABOUTBOX)
  {
    CTabAboutDlg dlgAbout;
```

0098

```
    dlgAbout.DoModal();
  }
  else
  {
    CDialog::OnSysCommand(nID, lParam);
  }
}

void CTabDialog::OnPaint()
{
//  CPaintDC dc(this); // device context for painting

  // TODO: Add your message handler code here

  // Do not call CDialog::OnPaint() for painting messages

  if (IsIconic())
  {
    CPaintDC dc(this); // device context for painting

    SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

    // Center icon in client rectangle
    int cxIcon = GetSystemMetrics(SM_CXICON);
    int cyIcon = GetSystemMetrics(SM_CYICON);
    CRect rect;
    GetClientRect(&rect);
    int x = (rect.Width() - cxIcon + 1) / 2;
    int y = (rect.Height() - cyIcon + 1) / 2;

    // Draw the icon
    dc.DrawIcon(x, y, m_hIcon);
  }
  else
  {
    //m_StatusBar.RedrawWindow();
    CDialog::OnPaint();
  }

}

void CTabDialog::OnMove(int x, int y)
{
  CDialog::OnMove(x, y);

  // TODO: Add your message handler code here

}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CTabDialog::OnQueryDragIcon()
{
  return (HCURSOR) m_hIcon;
}

HBRUSH CTabDialog::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{

  HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);

  // TODO: Change any attributes of the DC here

  // TODO: Return a different brush if the default is not desired
  return hbr;
#if 0
  return m_brush;
#endif
}

void CTabDialog::OnOpenMenu()
{
```

0099

```
  // TODO: Add your command handler code here
//  char BASED_CODE szFilter[] = "Text Files (*.txt)|*.txt||";
//  CFileDialog fileDlg(TRUE,NULL,NULL,0L,szFilter,NULL);


//  CFileDialog fileDlg(TRUE);
  CFileDialog fileDlg(TRUE,"txt",NULL,OFN_HIDEREADONLY,"Text Files (*.txt)|*.txt|All Files (*.*)|*.*||",this)

  if( fileDlg.DoModal() == IDOK )
    {
      int array_size = 0;
      CString string;
      CStringArray array;
      CStdioFile file(fileDlg.GetPathName(),CFile::modeRead | CFile::typeText);
      while( file.ReadString(string) )
      {
        array.SetAtGrow(array_size, string );
        array_size++;
      }
      g_DSPManager->SetDSPSettings(array);
      SetWindowText(fileDlg.GetFileName());
      // MM 5/13/99 Invalidate the window so that window is redrawn
      Invalidate(false);
    }

}

void CTabDialog::OnSaveMenu()
{

  // TODO: Add your command handler code here

  CFileDialog fileDlg(FALSE,"txt",NULL,OFN_OVERWRITEPROMPT|OFN_HIDEREADONLY,"Text Files (*.txt)|*.txt||",this
;

  if( fileDlg.DoModal() == IDOK )
    {
      int array_index = 0;
      CString string;
      CStringArray array;
      g_DSPManager->GetDSPSettings(array);
      CStdioFile file(fileDlg.GetPathName(),CFile::modeCreate | CFile::modeWrite | CFile::typeText);
      while( array_index < array.GetSize() )
      {
        string = array.GetAt(array_index);
        file.WriteString(string);
        file.WriteString("\n");
        array_index++;
      }
    }

}

void CTabDialog::OnFileExportparamfile()
{

  CFileDialog fileDlg(FALSE,"prm",NULL,OFN_OVERWRITEPROMPT|OFN_HIDEREADONLY,"Parameter Files (*.prm)|*.prm||",
this);

  if( fileDlg.DoModal() == IDOK )
    {
      int array_index = 0;
      CString string;
      CStringArray array;
      g_DSPManager->GetFilterBlob(array);
      CStdioFile file(fileDlg.GetPathName(),CFile::modeCreate | CFile::modeWrite | CFile::typeBinary);
      while( array_index < array.GetSize() )
      {
        string = array.GetAt(array_index);
        file.WriteString(string);
        file.WriteString("\n");
        array_index++;
```

0100

```
    }
  }
}

void CTabDialog::OnTimer(UINT nIDEvent)
{
  // TODO: Add your message handler code here and/or call default
  if( nIDEvent == TIMERID )
    {
      DisplayMode(g_DSPManager->GetHDCDMode());
      DisplayI2CState(0L);
    }
  else
    {
      CDialog::OnTimer(nIDEvent);
    }


}


void CTabDialog::DisplayMode(int value)
{

  if( value < 0 )
    {
      DisplayI2CState(-value);
      /* char s[100];
      sprintf(s,"0x%x",-value);
      m_StatusBar.GetStatusBarCtrl().SetText(s,3,0); */
      /* if( m_timerID )
        KillTimer( m_timerID );
      CI2CCommErrorDialog dlg;
      char s[100];
      sprintf(s,"0x%x",-value);
      dlg.m_ErrorCode = s;
      if( dlg.DoModal() == IDCANCEL )
        EndDialog(IDCANCEL);
      else
        m_timerID = SetTimer(TIMERID,50,NULL);  */
    }
  else if( value )
    {
      if( !m_HDCDDisplay )
      {
        m_ICnt = 0;
        m_AIdx = 0;
        m_StatusBar.GetStatusBarCtrl().SetText(Detect_Array[m_AIdx++],1,0);
        //SetWindowText(Detect_Array[m_AIdx++]);
      }
      else
      {
        m_ICnt++;
        if( m_ICnt > 5 )
        {
          m_ICnt = 0;
          m_StatusBar.GetStatusBarCtrl().SetText(Detect_Array[m_AIdx++],1,0);
          //SetWindowText(Detect_Array[m_AIdx++]);
          if( m_AIdx >= AARRAYSIZE )
            m_AIdx = 0;
        }
      }
      m_HDCDDisplay = true;
    }
  else
    {
      if( m_HDCDDisplay )
        m_StatusBar.GetStatusBarCtrl().SetText("BAD CD..NO DONUT",1,0);
        //SetWindowText("BAD CD .. NO DONUT");
      m_HDCDDisplay = false;
    }
```

0101

```
}


void CTabDialog::SetStatusString(int which, CString & s)
{

  m_StatusBar.GetStatusBarCtrl().SetText(s,0,0);

}

void CTabDialog::DisplayI2CState(long error)
{

  if( error )
  {
    char s[100];
    sprintf(s,"0x%x",error);
    m_StatusBar.GetStatusBarCtrl().SetText(s,3,0);
    m_ErrorCounter = 100;    // 5 seconds
  }
  else
  {
    if( m_ErrorCounter )
    {
      m_ErrorCounter--;
      if( m_ErrorCounter > 0 )
        return;
    }   if( g_DSPManager->IsBusy() )
      {
        m_StatusBar.GetStatusBarCtrl().SetText("Transmitting..",3,0);
      }
    else
      {
        m_StatusBar.GetStatusBarCtrl().SetText("",3,0);
      }
  }

}
```

0102

```
#if !defined(AFX_STWAVEREJPAGE_H__0CF7E209_D790_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_STWAVEREJPAGE_H__0CF7E209_D790_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// StWaveRejPage.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CStWaveRejPage dialog

class CTabDialog;

class CStWaveRejPage : public CPropertyPage
{
  DECLARE_DYNCREATE(CStWaveRejPage)

// Construction
public:
  CStWaveRejPage();
  ~CStWaveRejPage();

// Dialog Data
  //{{AFX_DATA(CStWaveRejPage)
  enum { IDD = IDD_PP5 };
  CButton m_BypassButton;
  CSliderCtrl m_Notch3BoostSlider;
  CSliderCtrl m_Notch3QSlider;
  CSliderCtrl m_Notch2CutSlider;
  CSliderCtrl m_Notch2QSlider;
  CSliderCtrl m_Notch2FreqSlider;
  CSliderCtrl m_Notch1CutSlider;
  CSliderCtrl m_Notch1QSlider;
  CSliderCtrl m_Notch1FreqSlider;
  //}}AFX_DATA


// Overrides
  // ClassWizard generate virtual function overrides
  //{{AFX_VIRTUAL(CStWaveRejPage)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  // Generated message map functions
  //{{AFX_MSG(CStWaveRejPage)
  virtual BOOL OnInitDialog();
  afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
  afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
  afx_msg void OnPaint();
  afx_msg void OnBypass();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()

  CTabDialog *m_ParentWindow;
  void SendStringToUI(int which);

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STWAVEREJPAGE_H__0CF7E209_D790_11D2_96EE_006097CDB9E2__INCLUDED_)
```

```
// StWaveRejPage.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "StWaveRejPage.h"
#include "TabDialog.h"
#include "DSP56kManager.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CStWaveRejPage property page

IMPLEMENT_DYNCREATE(CStWaveRejPage, CPropertyPage)

CStWaveRejPage::CStWaveRejPage() : CPropertyPage(CStWaveRejPage::IDD)
{
  //{{AFX_DATA_INIT(CStWaveRejPage)
  //}}AFX_DATA_INIT
}

CStWaveRejPage::~CStWaveRejPage()
{
}

void CStWaveRejPage::DoDataExchange(CDataExchange* pDX)
{
  CPropertyPage::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CStWaveRejPage)
  DDX_Control(pDX, IDC_CHECK5, m_BypassButton);
  DDX_Control(pDX, IDC_SLIDER9, m_Notch3BoostSlider);
  DDX_Control(pDX, IDC_SLIDER8, m_Notch3QSlider);
  DDX_Control(pDX, IDC_SLIDER6, m_Notch2CutSlider);
  DDX_Control(pDX, IDC_SLIDER5, m_Notch2QSlider);
  DDX_Control(pDX, IDC_SLIDER4, m_Notch2FreqSlider);
  DDX_Control(pDX, IDC_SLIDER3, m_Notch1CutSlider);
  DDX_Control(pDX, IDC_SLIDER2, m_Notch1QSlider);
  DDX_Control(pDX, IDC_SLIDER1, m_Notch1FreqSlider);
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CStWaveRejPage, CPropertyPage)
  //{{AFX_MSG_MAP(CStWaveRejPage)
  ON_WM_VSCROLL()
  ON_WM_SHOWWINDOW()
  ON_WM_PAINT()
  ON_BN_CLICKED(IDC_CHECK5, OnBypass)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CStWaveRejPage message handlers

BOOL CStWaveRejPage::OnInitDialog()
{
  CPropertyPage::OnInitDialog();

  // TODO: Add extra initialization here
  m_Notch1FreqSlider.SetRange(0,CONTROL_RANGE);
  m_Notch1FreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch3Freq));
  m_Notch1FreqSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_Notch1FreqSlider.SetPageSize(PG_CONTROL_AMT);

  m_Notch1QSlider.SetRange(0,CONTROL_RANGE);
  m_Notch1QSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch3Q));
  m_Notch1QSlider.SetTicFreq((CONTROL_RANGE+1)/16);
```

0104

```
  m_Notch1QSlider.SetPageSize(PG_CONTROL_AMT);

  m_Notch1CutSlider.SetRange(0,CONTROL_RANGE);
  m_Notch1CutSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch3Cut));
  m_Notch1CutSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_Notch1CutSlider.SetPageSize(PG_CONTROL_AMT);

  m_Notch2FreqSlider.SetRange(0,CONTROL_RANGE);
  m_Notch2FreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch4Freq));
  m_Notch2FreqSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_Notch2FreqSlider.SetPageSize(PG_CONTROL_AMT);

  m_Notch2QSlider.SetRange(0,CONTROL_RANGE);
  m_Notch2QSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch4Q));
  m_Notch2QSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_Notch2QSlider.SetPageSize(PG_CONTROL_AMT);

  m_Notch2CutSlider.SetRange(0,CONTROL_RANGE);
  m_Notch2CutSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch4Cut));
  m_Notch2CutSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_Notch2CutSlider.SetPageSize(PG_CONTROL_AMT);

  m_Notch3QSlider.SetRange(0,CONTROL_RANGE);
  m_Notch3QSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch5Q));
  m_Notch3QSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_Notch3QSlider.SetPageSize(PG_CONTROL_AMT);

  m_Notch3BoostSlider.SetRange(0,CONTROL_RANGE);
  m_Notch3BoostSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch5Cut));
  m_Notch3BoostSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_Notch3BoostSlider.SetPageSize(PG_CONTROL_AMT);


  m_ParentWindow = (CTabDialog *) GetParent()->GetParent();

  return TRUE;  // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CStWaveRejPage::SendStringToUI(int which)
{

    CString str;

    g_DSPManager->GetStringValue(which,str);
    m_ParentWindow->SetStatusString(0,str);

}

void CStWaveRejPage::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
  // TODO: Add your message handler code here and/or call default
  CSliderCtrl *slider = (CSliderCtrl *)pScrollBar;
  int which;

  CPropertyPage::OnVScroll(nSBCode, nPos, pScrollBar);

  Sleep(50);
  if( slider == &m_Notch1FreqSlider )
    which = kNotch3Freq;
  else if( slider == &m_Notch1QSlider )
    which = kNotch3Q;
  else if( slider == &m_Notch1CutSlider )
    which = kNotch3Cut;
  else if( slider == &m_Notch2FreqSlider )
    which = kNotch4Freq;
  else if( slider == &m_Notch2QSlider )
    which = kNotch4Q;
  else if( slider == &m_Notch2CutSlider )
    which = kNotch4Cut;
  else if( slider == &m_Notch3QSlider )
    which = kNotch5Q;
```

```
  else if( slider == &m_Notch3BoostSlider )
    which = kNotch5Cut;
  else
    return;

  g_DSPManager->SetParamValue(CONTROL_RANGE-slider->GetPos(),which);
  SendStringToUI(which);

}

void CStWaveRejPage::OnShowWindow(BOOL bShow, UINT nStatus)
{
  CPropertyPage::OnShowWindow(bShow, nStatus);

  // TODO: Add your message handler code here

}

void CStWaveRejPage::OnPaint()
{
  CPaintDC dc(this); // device context for painting

  // TODO: Add your message handler code here
  m_Notch1FreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch3Freq));
  m_Notch1QSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch3Q));
  m_Notch1CutSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch3Cut));

  m_Notch2FreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch4Freq));
  m_Notch2QSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch4Q));
  m_Notch2CutSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch4Cut));

  m_Notch3QSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch5Q));
  m_Notch3BoostSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kNotch5Cut));

  m_BypassButton.SetCheck(g_DSPManager->GetBypassSection(kBypassConecry));
  // Do not call CPropertyPage::OnPaint() for painting messages
}

void CStWaveRejPage::OnBypass()
{
  // TODO: Add your control notification handler code here
  int state = m_BypassButton.GetState() & 0x3;

  g_DSPManager->SetBypassSection(state,kBypassConecry);

}
```

```
/*

  There a number of blocks (processing units).
  The signal flow is linear currently.
  Each item is an ascii line of text
  The file format is as follows:

  version number

  block type
  number of params for block
  block param #1
  block param #2
  block param #3
  ...
  block param #n

  block type
  number of params for block
  block param #1
  block param #2
  block param #3
  ...
  block param #n

  ...

 */

// file version number
#define BLOB_FILE_VERSION 1

// block type descriptor
typedef enum {
    BLOCK_DELAY,
    BLOCK_GAIN,
    BLOCK_PEQ,
    BLOCK_AP,
    BLOCK_LS,
    BLOCK_HS,
    BLOCK_HP,
    BLOCK_LP,
    BLOCK_LP2
} ProcessType;
```

0107

```
// SPIPEMicroComm.h: interface for the SPIPEMicroComm class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_SPIPEMICROCOMM_H__33F9EF06_F6EF_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_SPIPEMICROCOMM_H__33F9EF06_F6EF_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "DSPComm.h"

class SPIPEMicroComm : public DSPComm
{

  int cmd_delay;
  int io_delay;

public:
  SPIPEMicroComm();
  virtual ~SPIPEMicroComm();
  virtual long SendDSPWord(long);
  virtual long SendDSPMemory(char *,long);


};

#endif // !defined(AFX_SPIPEMICROCOMM_H__33F9EF06_F6EF_11D2_96EE_006097CDB9E2__INCLUDED_)
```

0108

```
// SPIPEMicroComm.cpp: implementation of the SPIPEMicroComm class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "sa.h"
#include "SPIPEMicroComm.h"

#include "unit_ppi.h"
#include "functs.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

SPIPEMicroComm::SPIPEMicroComm()
{

  cmd_delay = 1000;
  io_delay = 50;

  if( load_dll()==false )
  {
    MessageBox(NULL,"UNIT_PPI not found in the system directory. Fix this and restart application.","Danger, I
anger Will Robinson", MB_OK | MB_ICONSTOP );
  }
  else
  {
    init_port_spi(1);
    set_cmd_delay_cnt_value(cmd_delay);
    set_io_delay_cnt_value(io_delay);
  }


}

SPIPEMicroComm::~SPIPEMicroComm()
{

}

long SPIPEMicroComm::SendDSPWord(long value)
{

  return( spi_xchng24(value) );

}

long SPIPEMicroComm::SendDSPMemory(char *data, long len)
{

  return( 0L );

}
```

0109

```
#if !defined(AFX_SHELVPAGE_H__6F151624_D84D_11D2_96EE_006097CDB9E2__INCLUDED_)
#define AFX_SHELVPAGE_H__6F151624_D84D_11D2_96EE_006097CDB9E2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ShelvPage.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CShelvPage dialog

class CTabDialog;

class CShelvPage : public CPropertyPage
{
  DECLARE_DYNCREATE(CShelvPage)

// Construction
public:
  CShelvPage();
  ~CShelvPage();

// Dialog Data
  //{{AFX_DATA(CShelvPage)
  enum { IDD = IDD_PP6 };
  CButton m_BypassSecondButton;
  CButton m_BypassFirstButton;
  CSliderCtrl m_HiBoostSlider;
  CSliderCtrl m_HiFreqSlider;
  CSliderCtrl m_LoBoostSlider;
  CSliderCtrl m_LoFreqSlider;
  //}}AFX_DATA


// Overrides
  // ClassWizard generate virtual function overrides
  //{{AFX_VIRTUAL(CShelvPage)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  // Generated message map functions
  //{{AFX_MSG(CShelvPage)
  virtual BOOL OnInitDialog();
  afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
  afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
  afx_msg void OnPaint();
  afx_msg void OnBypassFirst();
  afx_msg void OnBypassSecond();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()

  CTabDialog *m_ParentWindow;
  void SendStringToUI(int which);

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SHELVPAGE_H__6F151624_D84D_11D2_96EE_006097CDB9E2__INCLUDED_)
```

0110

```
// ShelvPage.cpp : implementation file
//

#include "stdafx.h"
#include "sa.h"
#include "ShelvPage.h"
#include "TabDialog.h"
#include "DSP56kManager.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CShelvPage property page

IMPLEMENT_DYNCREATE(CShelvPage, CPropertyPage)

CShelvPage::CShelvPage() : CPropertyPage(CShelvPage::IDD)
{
  //{{AFX_DATA_INIT(CShelvPage)
  //}}AFX_DATA_INIT
}

CShelvPage::~CShelvPage()
{
}

void CShelvPage::DoDataExchange(CDataExchange* pDX)
{
  CPropertyPage::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CShelvPage)
  DDX_Control(pDX, IDC_CHECK3, m_BypassSecondButton);
  DDX_Control(pDX, IDC_CHECK1, m_BypassFirstButton);
  DDX_Control(pDX, IDC_SLIDER6, m_HiBoostSlider);
  DDX_Control(pDX, IDC_SLIDER4, m_HiFreqSlider);
  DDX_Control(pDX, IDC_SLIDER3, m_LoBoostSlider);
  DDX_Control(pDX, IDC_SLIDER1, m_LoFreqSlider);
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CShelvPage, CPropertyPage)
  //{{AFX_MSG_MAP(CShelvPage)
  ON_WM_VSCROLL()
  ON_WM_SHOWWINDOW()
  ON_WM_PAINT()
  ON_BN_CLICKED(IDC_CHECK1, OnBypassFirst)
  ON_BN_CLICKED(IDC_CHECK3, OnBypassSecond)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CShelvPage message handlers

BOOL CShelvPage::OnInitDialog()
{
  CPropertyPage::OnInitDialog();

  // TODO: Add extra initialization here
  m_LoFreqSlider.SetRange(0,CONTROL_RANGE);
  m_LoFreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kLoShelfFreq));
  m_LoFreqSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_LoFreqSlider.SetPageSize(PG_CONTROL_AMT);

  m_LoBoostSlider.SetRange(0,CONTROL_RANGE);
  m_LoBoostSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kLoShelfGain));
  m_LoBoostSlider.SetTicFreq((CONTROL_RANGE+1)/16);
  m_LoBoostSlider.SetPageSize(PG_CONTROL_AMT);
```

```
    m_HiFreqSlider.SetRange(0,CONTROL_RANGE);
    m_HiFreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kHiShelfFreq));
    m_HiFreqSlider.SetTicFreq((CONTROL_RANGE+1)/16);
    m_HiFreqSlider.SetPageSize(PG_CONTROL_AMT);

    m_HiBoostSlider.SetRange(0,CONTROL_RANGE);
    m_HiBoostSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kHiShelfGain));
    m_HiBoostSlider.SetTicFreq((CONTROL_RANGE+1)/16);
    m_HiBoostSlider.SetPageSize(PG_CONTROL_AMT);


    m_ParentWindow = (CTabDialog *) GetParent()->GetParent();

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CShelvPage::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
  // TODO: Add your message handler code here and/or call default
  CSliderCtrl *slider = (CSliderCtrl *)pScrollBar;
  int which;

  CPropertyPage::OnVScroll(nSBCode, nPos, pScrollBar);

  Sleep(50);
  if( slider == &m_LoFreqSlider )
    which = kLoShelfFreq;
  else if( slider == &m_LoBoostSlider )
    which = kLoShelfGain;
  else if( slider == &m_HiFreqSlider )
    which = kHiShelfFreq;
  else if( slider == &m_HiBoostSlider )
    which = kHiShelfGain;
  else
    return;

  g_DSPManager->SetParamValue(CONTROL_RANGE-slider->GetPos(),which);
  SendStringToUI(which);


}

void CShelvPage::SendStringToUI(int which)
{

    CString str;

    g_DSPManager->GetStringValue(which,str);
    m_ParentWindow->SetStatusString(0,str);

}

void CShelvPage::OnShowWindow(BOOL bShow, UINT nStatus)
{
  CPropertyPage::OnShowWindow(bShow, nStatus);

  // TODO: Add your message handler code here

}

void CShelvPage::OnPaint()
{
  CPaintDC dc(this); // device context for painting

  // TODO: Add your message handler code here
  m_LoFreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kLoShelfFreq));
  m_LoBoostSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kLoShelfGain));
  m_HiFreqSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kHiShelfFreq));
  m_HiBoostSlider.SetPos(CONTROL_RANGE-g_DSPManager->GetParamValue(kHiShelfGain));

  m_BypassFirstButton.SetCheck(g_DSPManager->GetBypassSection(kBypassLoshelf));
```

0112

```
  m_BypassSecondButton.SetCheck(g_DSPManager->GetBypassSection(kBypassHiShelf));
  // Do not call CPropertyPage::OnPaint() for painting messages
}

void CShelvPage::OnBypassFirst()
{
  // TODO: Add your control notification handler code here
  int state = m_BypassFirstButton.GetState() & 0x3;

  g_DSPManager->SetBypassSection(state,kBypassLoshelf);

}

void CShelvPage::OnBypassSecond()
{
  // TODO: Add your control notification handler code here
  int state = m_BypassSecondButton.GetState() & 0x3;

  g_DSPManager->SetBypassSection(state,kBypassHiShelf);

}
```

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

#ifdef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
    "#define _AFX_NO_OLE_RESOURCES\r\n"
    "#define _AFX_NO_TRACKER_RESOURCES\r\n"
    "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
    "\r\n"
    "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\r\n"
    "#ifdef _WIN32\r\n"
    "LANGUAGE 9, 1\r\n"
    "#pragma code_page(1252)\r\n"
    "#endif\r\n"
    "#include ""res\\sa.rc2""  // non-Microsoft Visual C++ edited resources\r\n"
    "#include ""afxres.rc""          // Standard components\r\n"
    "#endif\0"
END

#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////////////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON    DISCARDABLE     "res\\sa.ico"

////////////////////////////////////////////////////////////////////////////////
//
// Dialog
```

0114

```
//

IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 259, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About KOJ's Amazing Town Crier"
FONT 8, "MS Sans Serif"
BEGIN
    ICON            IDR_MAINFRAME,IDC_STATIC,11,17,21,20
    LTEXT           "KOJ's Amazing Town Crier Version 1.0",IDC_STATIC,40,10,
                    119,8,SS_NOPREFIX
    LTEXT           "Copyright (C) PMI 1999",IDC_STATIC,40,25,119,8
    DEFPUSHBUTTON   "OK",IDOK,220,7,32,14,WS_GROUP
END

IDD_SA_DIALOG DIALOGEX 0, 0, 229, 175
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_APPWINDOW
CAPTION "SuperAudio"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,172,7,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,172,23,50,14
    PUSHBUTTON      "Poke Sequence #1",IDC_BUTTON1,35,83,109,25
    PUSHBUTTON      "Poke Sequence #2",IDC_BUTTON2,33,123,115,28
END

IDD_DIALOG1 DIALOG DISCARDABLE  0, 0, 236, 229
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "KOJ's Amazing Town Crier"
MENU IDR_MENU1
FONT 8, "MS Sans Serif"
BEGIN
END

IDD_PP1 DIALOG DISCARDABLE  0, 0, 195, 127
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Test"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "Send SPI Sequence #1",IDC_BUTTON1,57,36,86,22
    PUSHBUTTON      "Send SPI Sequence #2",IDC_BUTTON2,55,81,93,31
END

IDD_PP2 DIALOG DISCARDABLE  0, 0, 288, 146
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Main"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Slider1",IDC_SLIDER1,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,58,25,22,86
    LTEXT           "Pre",IDC_STATIC,28,117,12,8
    CONTROL         "Bypass Processing",IDC_CHECK1,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,173,57,76,10
    CONTROL         "Slider1",IDC_SLIDER3,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,27,25,22,86
    LTEXT           "Post",IDC_STATIC,57,117,15,8
    GROUPBOX        "Volume",IDC_STATIC,19,15,135,118
    CONTROL         "Bypass HDCD",IDC_CHECK2,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,173,74,76,10
    CONTROL         "Bypass HDCD Gain Scale",IDC_CHECK3,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,173,91,98,10
    PUSHBUTTON      "Reset All",IDC_BUTTON1,173,31,50,14
    CONTROL         "Analog Input",IDC_CHECK4,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,173,108,56,10
    CONTROL         "Slider1",IDC_SLIDER7,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,89,25,22,86
    LTEXT           "Analog",IDC_STATIC,85,117,23,8
    CONTROL         "Slider1",IDC_SLIDER8,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,119,25,22,86
    LTEXT           "Delay",IDC_STATIC,117,117,19,8
END
```

0115

```
IDD_PP3 DIALOG DISCARDABLE  0, 0, 195, 127
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Compression"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Bypass Compression",IDC_CHECK1,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,29,24,104,10
END

IDD_PP4 DIALOG DISCARDABLE  0, 0, 244, 159
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Resonance Compensation"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Slider1",IDC_SLIDER1,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,20,24,20,85
    CONTROL         "Slider1",IDC_SLIDER2,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,48,24,21,85
    CONTROL         "Slider1",IDC_SLIDER3,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,76,24,20,85
    LTEXT           "Freq.",IDC_STATIC,17,113,20,11
    LTEXT           "Q",IDC_STATIC,51,113,8,11
    LTEXT           "Cut/Boost",IDC_STATIC,71,113,33,11
    GROUPBOX        "First Notch",IDC_STATIC,11,15,98,111
    CONTROL         "Slider1",IDC_SLIDER4,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,137,24,24,85
    CONTROL         "Slider1",IDC_SLIDER5,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,165,24,21,85
    CONTROL         "Slider1",IDC_SLIDER6,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,193,24,22,85
    LTEXT           "Freq.",IDC_STATIC,135,113,20,11
    LTEXT           "Q",IDC_STATIC,169,113,8,11
    LTEXT           "Cut/Boost",IDC_STATIC,189,113,32,11
    GROUPBOX        "Second Notch",IDC_STATIC,129,15,98,111
    CONTROL         "Bypass",IDC_CHECK5,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,33,136,39,10
    CONTROL         "Bypass",IDC_CHECK6,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,153,136,39,10
END

IDD_PP5 DIALOG DISCARDABLE  0, 0, 339, 162
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Standing Wave Rejection"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Slider1",IDC_SLIDER1,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,20,24,20,85
    CONTROL         "Slider1",IDC_SLIDER2,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,48,24,21,85
    CONTROL         "Slider1",IDC_SLIDER3,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,76,24,20,85
    LTEXT           "Freq.",IDC_STATIC,17,113,20,11
    LTEXT           "Q",IDC_STATIC,51,113,8,11
    LTEXT           "Cut/Boost",IDC_STATIC,70,113,35,11
    GROUPBOX        "First Notch",IDC_STATIC,11,15,98,111
    CONTROL         "Slider1",IDC_SLIDER4,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,229,24,24,85
    CONTROL         "Slider1",IDC_SLIDER5,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,257,24,21,85
    CONTROL         "Slider1",IDC_SLIDER6,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,285,24,22,85
    LTEXT           "Freq.",IDC_STATIC,227,113,20,11
    LTEXT           "Q",IDC_STATIC,261,113,8,11
    LTEXT           "Cut/Boost",IDC_STATIC,280,113,34,11
    GROUPBOX        "Second Notch",IDC_STATIC,221,15,98,111
    CONTROL         "Slider1",IDC_SLIDER8,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,140,25,21,85
    CONTROL         "Slider1",IDC_SLIDER9,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,168,25,22,85
    LTEXT           "Q",IDC_STATIC,144,114,8,11
    LTEXT           "Cut/Boost",IDC_STATIC,163,114,34,11
    GROUPBOX        "Boost Compensation",IDC_STATIC,125,15,77,111
```

0116

```
    CONTROL         "Bypass",IDC_CHECK5,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,146,137,41,10
END

IDD_PP6 DIALOG DISCARDABLE  0, 0, 187, 162
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Smiley-Face"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Slider1",IDC_SLIDER1,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,20,24,20,85
    CONTROL         "Slider1",IDC_SLIDER3,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,50,24,20,85
    LTEXT           "Freq.",-1,17,113,20,11
    LTEXT           "Cut/Boost",-1,43,113,34,11
    GROUPBOX        "Lo-Shelf",-1,11,15,70,111
    CONTROL         "Slider1",IDC_SLIDER4,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,108,24,24,85
    CONTROL         "Slider1",IDC_SLIDER6,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,139,24,22,85
    LTEXT           "Freq.",-1,106,113,20,11
    LTEXT           "Cut/Boost",-1,134,113,33,11
    GROUPBOX        "Hi-Shelf",-1,100,15,69,111
    CONTROL         "Bypass",IDC_CHECK1,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,25,136,37,12
    CONTROL         "Bypass",IDC_CHECK3,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,115,136,37,12
END

IDD_PP7 DIALOG DISCARDABLE  0, 0, 242, 162
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Cutoff Response"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Slider1",IDC_SLIDER1,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,20,24,20,85
    CONTROL         "Slider1",IDC_SLIDER2,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,48,24,21,85
    CONTROL         "Slider1",IDC_SLIDER3,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,76,24,20,85
    LTEXT           "Freq.",IDC_STATIC,17,113,20,11
    LTEXT           "Q",IDC_STATIC,51,113,8,11
    LTEXT           "Cut/Boost",IDC_STATIC,71,113,33,11
    GROUPBOX        "Low Cutoff",IDC_STATIC,11,15,98,111
    CONTROL         "Slider1",IDC_SLIDER4,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,137,24,24,85
    CONTROL         "Slider1",IDC_SLIDER5,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,165,24,21,85
    CONTROL         "Slider1",IDC_SLIDER6,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,193,24,22,85
    LTEXT           "Freq.",IDC_STATIC,135,113,20,11
    LTEXT           "Q",IDC_STATIC,169,113,8,11
    LTEXT           "Cut/Boost",IDC_STATIC,189,113,32,11
    GROUPBOX        "High Cutoff",IDC_STATIC,129,15,98,111
    CONTROL         "Bypass",IDC_CHECK5,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,39,139,41,12
    CONTROL         "Bypass",IDC_CHECK6,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,154,139,41,12
END

IDD_PP8 DIALOG DISCARDABLE  0, 0, 285, 164
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Cutoff Response"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Slider1",IDC_SLIDER1,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,20,24,20,85
    CONTROL         "Slider1",IDC_SLIDER2,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,48,24,21,85
    LTEXT           "Freq.",-1,17,113,20,11
    LTEXT           "Q",-1,51,113,8,11
    GROUPBOX        "Low Cutoff",-1,11,15,71,111
```

```
    CONTROL         "Slider1",IDC_SLIDER4,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,112,24,24,85
    CONTROL         "Slider1",IDC_SLIDER5,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,140,24,21,85
    LTEXT           "Freq.",-1,110,113,20,11
    LTEXT           "Q",-1,144,113,8,11
    GROUPBOX        "High Cutoff",-1,104,15,69,111
    CONTROL         "Bypass",IDC_CHECK4,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,25,136,40,13
    CONTROL         "Bypass",IDC_CHECK5,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,116,136,40,13
    CONTROL         "Slider1",IDC_SLIDER8,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,203,25,24,85
    CONTROL         "Slider1",IDC_SLIDER9,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,231,25,21,85
    LTEXT           "Freq.",-1,201,114,20,11
    LTEXT           "Q",-1,235,114,8,11
    GROUPBOX        "New High Cutoff",-1,195,15,69,111
    CONTROL         "Bypass",IDC_CHECK7,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,207,137,40,13
END

IDD_PP9 DIALOG DISCARDABLE  0, 0, 244, 164
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Resonance Compensation #2"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Slider1",IDC_SLIDER1,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,20,24,20,85
    CONTROL         "Slider1",IDC_SLIDER2,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,48,24,21,85
    CONTROL         "Slider1",IDC_SLIDER3,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,76,24,20,85
    LTEXT           "Freq.",IDC_STATIC,17,113,20,11
    LTEXT           "Q",IDC_STATIC,51,113,8,11
    LTEXT           "Cut/Boost",IDC_STATIC,71,113,33,11
    GROUPBOX        "Third Notch",IDC_STATIC,11,15,98,111
    CONTROL         "Slider1",IDC_SLIDER4,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,137,24,24,85
    CONTROL         "Slider1",IDC_SLIDER5,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,165,24,21,85
    CONTROL         "Slider1",IDC_SLIDER6,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,193,24,22,85
    LTEXT           "Freq.",IDC_STATIC,135,113,20,11
    LTEXT           "Q",IDC_STATIC,169,113,8,11
    LTEXT           "Cut/Boost",IDC_STATIC,189,113,32,11
    GROUPBOX        "Fourth Notch",IDC_STATIC,129,15,98,111
    CONTROL         "Bypass",IDC_CHECK5,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,36,139,40,12
    CONTROL         "Bypass",IDC_CHECK6,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,155,139,40,12
END

IDD_DIALOG2 DIALOG DISCARDABLE  0, 0, 175, 66
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "Cancel",IDCANCEL,59,34,50,14
    LTEXT           "Initializing I2C.  Please Wait.....",IDC_STATIC,43,17,
                    98,8
END

IDD_DIALOG3 DIALOG DISCARDABLE  0, 0, 186, 132
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,107,95,50,14
    CONTROL         "COM1",IDC_RADIO1,"Button",BS_AUTORADIOBUTTON,23,57,36,
                    10
    CONTROL         "COM2",IDC_RADIO2,"Button",BS_AUTORADIOBUTTON,23,70,36,
```

```
                    10
    CONTROL         "COM3",IDC_RADIO3,"Button",BS_AUTORADIOBUTTON,23,82,36,
                    10
    CONTROL         "COM4",IDC_RADIO4,"Button",BS_AUTORADIOBUTTON,23,95,36,
                    10
    LTEXT           "Choose COM port and reset DSP board",IDC_STATIC,28,21,
                    125,8
    GROUPBOX        "COM Port",IDC_STATIC,15,45,68,69
END

IDD_DIALOG4 DIALOG DISCARDABLE  0, 0, 186, 121
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,27,72,50,14
    PUSHBUTTON      "Quit App",IDC_BUTTON1,101,72,48,14
    LTEXT           "I2C communication error!",IDC_STATIC,49,25,79,8
    EDITTEXT        IDC_EDIT1,49,42,68,12,ES_AUTOHSCROLL
END

IDD_PP10 DIALOG DISCARDABLE  0, 0, 187, 162
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Allpass"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Slider1",IDC_SLIDER1,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,20,24,20,85
    CONTROL         "Slider1",IDC_SLIDER3,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,50,24,20,85
    LTEXT           "Freq.",-1,17,113,20,11
    LTEXT           "Q",-1,53,113,8,8
    GROUPBOX        "Allpass",-1,11,15,70,111
    CONTROL         "Bypass",IDC_CHECK1,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,25,136,37,12
END

IDD_PP11 DIALOG DISCARDABLE  0, 0, 244, 159
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Double-Tuned Notch"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Slider1",IDC_SLIDER1,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,20,24,20,85
    CONTROL         "Slider1",IDC_SLIDER2,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,48,24,21,85
    LTEXT           "Freq.",-1,17,113,20,11
    LTEXT           "Q",-1,51,113,8,11
    GROUPBOX        "Notch",-1,11,15,70,111
    CONTROL         "Slider1",IDC_SLIDER5,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,109,24,21,85
    CONTROL         "Slider1",IDC_SLIDER6,"msctls_trackbar32",TBS_AUTOTICKS |
                    TBS_VERT | WS_TABSTOP,137,24,22,85
    LTEXT           "Q",-1,113,113,8,11
    LTEXT           "Cut/Boost",-1,133,113,32,11
    GROUPBOX        "Compensation",-1,99,15,71,111
    CONTROL         "Bypass",IDC_CHECK5,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,73,136,39,10
END


#ifndef _MAC
/////////////////////////////////////////////////////////////////////////////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 0,7,4,0
 PRODUCTVERSION 0,7,4,0
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
```

```
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
            VALUE "CompanyName", "Pacific Microsonics Inc.\0"
            VALUE "FileDescription", "sa MFC Application\0"
            VALUE "FileVersion", "0, 7, 4, 0\0"
            VALUE "InternalName", "sa\0"
            VALUE "LegalCopyright", "Copyright (C) PMI 1999\0"
            VALUE "OriginalFilename", "sa.EXE\0"
            VALUE "ProductName", "sa Application\0"
            VALUE "ProductVersion", "0, 7, 4, 0\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END

#endif    // !_MAC


/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 252
        TOPMARGIN, 7
        BOTTOMMARGIN, 48
    END

    IDD_SA_DIALOG, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 222
        TOPMARGIN, 7
        BOTTOMMARGIN, 168
    END

    IDD_DIALOG1, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 229
        TOPMARGIN, 7
        BOTTOMMARGIN, 222
    END

    IDD_PP1, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 188
        TOPMARGIN, 7
        BOTTOMMARGIN, 120
    END

    IDD_PP2, DIALOG
```

```
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 281
    TOPMARGIN, 7
    BOTTOMMARGIN, 139
END

IDD_PP3, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 188
    TOPMARGIN, 7
    BOTTOMMARGIN, 120
END

IDD_PP4, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 237
    TOPMARGIN, 7
    BOTTOMMARGIN, 152
END

IDD_PP5, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 332
    TOPMARGIN, 7
    BOTTOMMARGIN, 155
END

IDD_PP6, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 180
    TOPMARGIN, 7
    BOTTOMMARGIN, 155
END

IDD_PP7, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 235
    TOPMARGIN, 7
    BOTTOMMARGIN, 155
END

IDD_PP8, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 278
    TOPMARGIN, 7
    BOTTOMMARGIN, 157
END

IDD_PP9, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 237
    TOPMARGIN, 7
    BOTTOMMARGIN, 157
END

IDD_DIALOG2, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 168
    TOPMARGIN, 7
    BOTTOMMARGIN, 59
END

IDD_DIALOG3, DIALOG
```

```
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 179
        TOPMARGIN, 7
        BOTTOMMARGIN, 125
    END

    IDD_DIALOG4, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 179
        TOPMARGIN, 7
        BOTTOMMARGIN, 114
    END

    IDD_PP10, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 180
        TOPMARGIN, 7
        BOTTOMMARGIN, 155
    END

    IDD_PP11, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 237
        TOPMARGIN, 7
        BOTTOMMARGIN, 152
    END
END
#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////////
//
// Menu
//

IDR_MENU1 MENU DISCARDABLE
BEGIN
    POPUP "File"
    BEGIN
        MENUITEM "Open",                        ID_MENUITEM32771
        MENUITEM "Save",                        ID_MENUITEM32772
        MENUITEM "Export Param File",           ID_FILE_EXPORTPARAMFILE
    END
END


////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE DISCARDABLE
BEGIN
    IDS_ABOUTBOX            "&About KOJ's Amazing Town Crier..."
END

#endif    // English (U.S.) resources
////////////////////////////////////////////////////////////////////////////


#ifndef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
```

```
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif
#include "res\sa.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#endif
/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

```
        opt     cc,cex,cre,loc,so,mu,mex

    DEFINE          FAST    '1'
    DEFINE          MS6362  '1'

TABLE_COUNTER   EQU 10
RUN_COUNTER     EQU 10
INIT_COUNTER    EQU 10

    org y(10):$100
    org p(10):$980

; ********************************************************
; ********************************************************

    SECTION     SUPER

; ********************************************************
; ********************************************************

;--------------------------------------------------------------
; HDCD detect and expand code
;--------------------------------------------------------------

        include 'detect.asm'
        include 'expgain.asm'

;*****************************************************************************
;
;   Adopted from Motorola sample code.
;
;*****************************************************************************
                page    132,60
                include 'ioequ.asm'
                include 'vectors.asm'
;               include 'mbiquad.asm'
                include 'mpeq.asm'
                include 'mshelv.asm'
                include 'mhipass.asm'
                include 'mlopass.asm'
                include 'allpass.asm'
                list

CONFIGURE_SPI   MACRO

;   move    #0,x0
;   move    x0,X:M_HSAR                 ; clear I2C address

    ; M_HCKR
    ; bits 13:12 HFM1:0 noise filter setting (off)
    ; bit 1 CPOL
    ; bit 0 CPHA
    ;
    ; 00 xxxx xxxx xx01
;   move    #1,x0
;   move    #0,x0
;   move    x0,X:M_HCKR
;   commented out code should yeild a value of 0 for M_HCKR.

    ; M_HCSR
    ; bit 13:12 HCSR HRIE
    ; bit 11 HCSR HTIE
    ; bit 10 HCSR HBIE
    ; bit 9 HCSR Idle (no care)
    ; bit 8:7 HRQE Host-request enable (asserted if ready to receive)
    ; bit 6 HMST Master mode (disabled)
    ; bit 5 HFIFO FIFO control (disabled)
    ; bit 4 HCKFR Clock freeze (off)
    ; bit 3:2 HM1:0 24 bit mode
    ; bit 1 HI2C Enables I2C mode
    ; bit 0 HEN Enable port
```

0124

```
    ;
    ; xx01 0010 000x 1001
    move    #$1209,x0
    move    x0,X:M_HCSR

    ; set command to known value
    move    #SPITXDEF,x0
    move    x0,x:spixmit
;   movep   x:spixmit,x:M_HTX

    ENDM

CONFIGURE_I2C   MACRO

    ; turn off port
    move    #0,x0
    move    x0,X:M_HCSR

    ; leave address alone
;   move    #0,x0
;   move    x0,X:M_HSAR                 ; clear I2C address

;   commented out code should yeild a value of 0 for M_HCKR.
    ; M_HCKR
    ; bits 13:12 HFM1:0 noise filter setting on
    ;
    ; 11 xxxx xxxx xx00
    move    #>$3000,x0
    move    x0,X:M_HCKR

    ; M_HCSR
    ; bit 13:12 HCSR HRIE
    ; bit 11 HCSR HTIE
    ; bit 10 HCSR HBIE
    ; bit 9 HCSR Idle (no care)
    ; bit 8:7 HRQE Host-request enable (asserted if ready to receive)
    ; bit 6 HMST Master mode (disabled)
    ; bit 5 HFIFO FIFO control (disabled)
    ; bit 4 HCKFR Clock freeze (off)
    ; bit 3:2 HM1:0 24 bit mode
    ; bit 1 HI2C Enables I2C mode
    ; bit 0 HEN Enable port
    ;
    ; xx01 1100 101x 1010
    move    #$1CAA,x0
    move    x0,X:M_HCSR

    ; set command to known value
    move    #0,x0
    move    x0,x:spixmit
;   movep   x:spixmit,x:M_HTX

    ENDM

    XDEF        fcontrol
    XREF        HDCD_DETECT_INIT,HDCD_DETECT,lsbcntl
    XREF        ltemp,rtemp
    XREF        dcontrol
    XREF        HDCD_GAIN_INIT,HDCD_DYNAMICS
    XREF        detect
    XREF        rmodel,bitcntl

SPITXDEF    equ     $444444

        org     xi:
DRX_BUFF_BASE   ds  2
ARX_BUFF_BASE   ds  2

TX_BUFF_BASE    ds  4       ; unprocessed followed by processed data

;RX_PTR     ds  1
TX_PTR      ds  1
```

0125

```
flags        ds  1
scnt         ds  1
RightReceive     equ     0
SPIReceive       equ     1
SPIOverrun       equ     2
SPIFIFOFull      equ     3
SPIUnderrun      equ     4
SPITransmit      equ     5
SPIStateBit      equ     6

spixmit      ds  1
spirecv      ds  1
shisaveA1    ds  1
shisaveX0    ds  1
shisaveR4    ds  1
shisaveN4    ds  1
NADA         ds  1
left         ds  1
right        ds  1
fcontrol     ds  1
indexData    ds  1
dpleft       ds  1
dpright      ds  1
stack        ds  20

; interleaved delay line
; independently located
    org x(20):$200
DELAYLEN     EQU 128                 ; 2.9 mS delay line
sdelayleft       dsm DELAYLEN        ; left
sdelayright      dsm DELAYLEN        ; right

    org yi:
YNADA        ds  1
volume       ds  1
prefader     ds  1
bypass       ds  1
hdcdbp       ds  1
gainscalebp  ds  1
ddxcompbp    ds  1
analogin     ds  1
avolume      ds  1
bypassmask   ds  1

BYPASS_NOTCH1    EQU 0
BYPASS_NOTCH2    EQU 1
BYPASS_NOTCH3    EQU 2
BYPASS_NOTCH4    EQU 3
BYPASS_HIPASS    EQU 4
BYPASS_LOPASS    EQU 5
BYPASS_LOSHELV   EQU 6
BYPASS_HISHELV   EQU 7
BYPASS_CONECRY   EQU 8
BYPASS_ALLPASS   EQU 9
BYPASS_DBNOTCH   EQU 10
BYPASS_NLOPASS   EQU 11

delayval     ds  1

; ****************************************
;   EQ/filter declarations
; ****************************************
; smiley face shelf EQ
    ; left channel
    DECLARE_LSH 1L,1.,-0.867
    DECLARE_HSH 2L,1.,-0.867

    ; right channel
    DECLARE_LSH 1R,1.,-0.867
    DECLARE_HSH 2R,1.,-0.867

; resonance compensation PEQ
```

```
    ; left channel
    DECLARE_PEQ 1L,1.,-0.9510565,0.7265425
    DECLARE_PEQ 2L,1.,-0.91,0.5

    DECLARE_PEQ 6L,1.,-0.9510565,0.7265425
    DECLARE_PEQ 7L,1.,-0.91,0.5

    ; right channel
    DECLARE_PEQ 1R,1.,-0.91,0.5
    DECLARE_PEQ 2R,1.,-0.91,0.5

    DECLARE_PEQ 6R,1.,-0.9510565,0.7265425
    DECLARE_PEQ 7R,1.,-0.91,0.5

; standing wave rejection PEQ
    ; left channel
    DECLARE_PEQ 3L,1.,-0.9999803,0.9937365
    DECLARE_PEQ 4L,1.,-0.91,0.5
    DECLARE_PEQ 5L,1.,-0.91,0.5

    ; right channel
    DECLARE_PEQ 3R,1.,-0.9999803,0.9937365
    DECLARE_PEQ 4R,1.,-0.91,0.5
    DECLARE_PEQ 5R,1.,-0.91,0.5

; hi and lo pass filters
    ; left
    DECLARE_HP  hpleft,0.0099733,0.25,0.9987285          ; 70 Hz
    DECLARE_LP  lpleft,1.1921856,0.6303886,1.4112871     ; 14000 Hz / q = 0.22

    ; right
    DECLARE_HP  hpright,0.0099733,0.25,0.9987285         ; 70 Hz
    DECLARE_LP  lpright,1.1921856,0.6303886,1.4112871    ; 14000 Hz

    ; new lopass
    ; left
    DECLARE_LP  lpleft2,1.,0.,0.
    ; right
    DECLARE_LP  lpright2,1.,0.,0.

; allpass  filters
    DECLARE_AP  apleft,-0.91,0.5
    DECLARE_AP  apright,-0.91,0.5


; Keith new notch with 2 gain around it.
    DECLARE_PEQ NT1L,1.,-0.9999803,0.9937365
    DECLARE_PEQ NT2L,1.,-0.91,0.5
    DECLARE_PEQ NT3L,1.,-0.91,0.5

    DECLARE_PEQ NT1R,1.,-0.9999803,0.9937365
    DECLARE_PEQ NT2R,1.,-0.91,0.5
    DECLARE_PEQ NT3R,1.,-0.91,0.5

;************************************************************************

        org     p:$100
START
main
        ori     #$03,mr                 ; mask interrupts
        movep   #$05000B,X:M_PCTL       ;
        move    #0,omr
        movec   #0,sp                   ; reset hardware stack pointer
;        movep   #$000003,x:M_IPRP      ; ESAI int's enabled and top Priority
        movep   #$000007,x:M_IPRP       ; SHI (1) and ESAI (2) int's enabled
        move    #stack,r6               ; initialize stack pointer
        move    #-1,m6                  ; linear addressing

        move    #0,x0
        move    x0,x:DRX_BUFF_BASE
        move    x0,x:DRX_BUFF_BASE+1
        move    x0,x:ARX_BUFF_BASE
```

```
move    x0,x:ARX_BUFF_BASE+1
move    x0,x:TX_BUFF_BASE
move    x0,x:TX_BUFF_BASE+1
move    x0,x:TX_BUFF_BASE+2
move    x0,x:TX_BUFF_BASE+3

clr     a
move    a,x:flags
move    a,x:scnt

; HDCD decoder initialize
move    #>1,x0
move    x0,x:fcontrol
move    #>16,x0
move    x0,x:lsbcntl
jsr     HDCD_DETECT_INIT
move    #>1,x0
move    x0,x:dcontrol
jsr     HDCD_GAIN_INIT

move    #-1,m0
move    #-1,m1
move    #-1,m2
move    #-1,m3
move    #-1,m4
move    #-1,m5
move    #-1,m6
move    #-1,m7

; set up volume and bypass switches
; output volume is set to zero so that filter parameters can
; be downloaded, followed by volume being set

move    #>$800000,x0
move    x0,y:volume
move    x0,y:prefader
move    x0,y:avolume
move    #>$7FFFFF,x0
move    x0,y:bypass
move    #>$0,x0
move    x0,y:hdcdbp
move    x0,y:gainscalebp
move    x0,y:ddxcompbp
move    x0,y:analogin
move    x0,y:bypassmask
move    x0,y:delayval
move    x0,x:indexData

move    #>sdelayleft,x0
move    x0,x:dpleft
move    #>sdelayright,x0
move    x0,x:dpright

; initialize EQ params
INIT_PEQ_PARAM 1L
INIT_PEQ_PARAM 1R
INIT_PEQ_PARAM 2L
INIT_PEQ_PARAM 2R
INIT_PEQ_PARAM 3L
INIT_PEQ_PARAM 3R
INIT_PEQ_PARAM 4L
INIT_PEQ_PARAM 4R
INIT_PEQ_PARAM 5L
INIT_PEQ_PARAM 5R
INIT_PEQ_PARAM 6L
INIT_PEQ_PARAM 6R
INIT_PEQ_PARAM 7L
INIT_PEQ_PARAM 7R

INIT_PEQ_PARAM  NT1L
INIT_PEQ_PARAM  NT2L
INIT_PEQ_PARAM  NT3L
```

0128

```
        INIT_PEQ_PARAM  NT1R
        INIT_PEQ_PARAM  NT2R
        INIT_PEQ_PARAM  NT3R

        INIT_SH_PARAM   1L
        INIT_SH_PARAM   1R
        INIT_SH_PARAM   2L
        INIT_SH_PARAM   2R

        INIT_LP_PARAM   lpleft
        INIT_LP_PARAM   lpright

        INIT_LP2_PARAM  lpleft2
        INIT_LP2_PARAM  lpright2

        INIT_AP_PARAM apleft
        INIT_AP_PARAM apright

            ; hi-pass does not need initializing as all params are downloaded




        ; setup serial host interface
        ;; CONFIGURE_SPI
        CONFIGURE_I2C

;-------------------------------------------------------------
; FST/FSR and SCKT/SCKR are generated from the PLD
; and fed to the DSP, A/D and D/A converters
;-------------------------------------------------------------

;put esai in reset state.
;       IF  @DEF('ANALOGIN')==0
        movep   #$000000,x:M_PCRC       ; MLS 12/20/97
        movep   #$000000,x:M_PRRC       ; MLS 12/20/97
;       ENDIF


        movep   #$0c0200,x:M_TCCR
                ;FST is input                                (bit22=0)
                ;external clock source drives SCKT          (bit21=0)
                ;negative FST polarity                      (bit19=1)
                ;data & FST clocked out on rising edge      (bit18=1)
                ;2 words per frame                          (bit13:9=00001)


        movep   #$0c0200,x:M_RCCR
                ;FSR is input                                (bit22=0)
                ;external clock source drives SCKR          (bit21=0)
                ;negative FSR polarity                      (bit19=1)
                ;data & FSR clocked in on rising edge       (bit18=0)BAK(121997)
                ;2 words per frame                          (bit13:9=00001)

        movep   #$000000,x:M_SAICR
```

```
        movep   #$d17D03,x:M_RCR
                ;RX1, RX0 enabled                           (bit1:0=11)
                ;RX2, RX3 disabled                          (bit3:2=00)
                ;reserved                                   (bit5:4=00)
                ;MSB shifted first                          (bit6=0)
                ;word left-aligned                          (bit7=0)
                ;network mode                               (bit9:8=01)
                ;32-bit slot length, 24-bit word length     (bit14:10=11111)
                ;word-length frame sync                     (bit15=0)
                ;frame sync occurs 1 clock cycle earlier    (bit16=1)
                ;reserved                                   (bit19:17=000)
                ;RLIE, RIE, REIE enabled                    (bit23:20=0101)
                ;bit23 RLIE
                ;bit22 RIE
                ;bit21 REDIE
                ;bit20 REIE

        movep   #$d13d00,x:M_TCR                            ;MLS 12/20/97
                ;TX0, TX1 enabled                           (bit3:0=0011)
                ;TX2, TX3, TX4, TX5 disabled                (bit5:4=00)
                ;MSB shifted first                          (bit6=0)
                ;word left-aligned                          (bit7=0)
                ;network mode                               (bit9:8=01)
                ;32-bit slot length, 24-bit word length     (bit14:10=11111)
                ;word length frame sync                     (bit15=0)
                ;frame sync occurs 1 clock cycle earlier    (bit16=0)
                ;reserved                                   (bit19:17=000)
                ;TLIE, TIE, TEIE enabled                    (bit23:20=0101)
                ;bit23 TLIE
                ;bit22 TIE
                ;bit21 TEDIE
                ;bit20 TEIE

        movep   #$000edb,x:M_PCRC       ; MLS 12/20/97
        movep   #$000edb,x:M_PRRC       ; MLS 12/20/97

        movep   #$ffffff,x:M_RSMA        ;MLS 12/20/97
        movep   #$ffffff,x:M_RSMB        ;MLS 12/20/97

        movep   #$000003,x:M_TSMA
        movep   #$000003,x:M_TSMB
        movep   #$000000,x:M_TX0        ;zero out transmitter 0
        movep   #$000000,x:M_TX1        ;zero out transmitter 1
        movep   #$000000,x:M_TX2        ;zero out transmitter 2
        movep   #$000000,x:M_TX3        ;zero out transmitter 3
        bset    #0,x:M_TCR              ;now enable TX0
        bset    #1,x:M_TCR              ;now enable TX1
        bset    #2,x:M_TCR              ;now enable TX2
        bset    #3,x:M_TCR              ;now enable TX3
        bset    #2,x:M_RCR              ;RE2 and TE3 must be set to enable
                                        ;TX3 and disable RX2

        ; turn on serial host interface
        bset    #0,X:M_HCSR

        andi    #$FC,mr                 ;enable all interrupt levels
                                        ;clear scaling bits

        bclr    #RightReceive,X:flags

;       jclr    #RightReceive,X:flags,*
;       bclr    #RightReceive,X:flags
;       jclr    #RightReceive,X:flags,*
;        move    #>RX_BUFF_BASE,x0
;        move    x0,x:RX_PTR
;        move    #>TX_BUFF_BASE,x0
;        move    x0,x:TX_PTR

;---------------------------------------------------------------
; Main loop
;---------------------------------------------------------------
LOOP
```

0130

```
        jclr    #RightReceive,X:flags,*
        bclr    #RightReceive,X:flags

        btst    #22,y:analogin
        .IF     <CC>
            move    x:DRX_BUFF_BASE,x0          ;receive left
            move    x:DRX_BUFF_BASE+1,x1        ;receive right
        .ELSE
            move    x:ARX_BUFF_BASE,x0          ;receive left
            move    x:ARX_BUFF_BASE+1,x1        ;receive right
        .ENDI

        ; store unprocessed (with volume) output first
        move    y:avolume,y0
        mpyr    y0,x0,a
        mpyr    y0,x1,b
        ; MM 5/20/99 Swapped channels
        tfr     x0,a    a,x:TX_BUFF_BASE+2      ;transmit left
        tfr     x1,b    b,x:TX_BUFF_BASE        ;transmit right

        ; write input into delay line
        move    #DELAYLEN-1,m0
        move    #DELAYLEN-1,m1
        move    x:dpleft,r0
        move    x:dpright,r1
        move    y:delayval,y0
        move    #>DELAYLEN,x0
        mpy     x0,y0,a     a,x:(r0)-
        move                b,x:(r1)-
        move    a,n0
        move    a,n1
        move                x:(r0+n0),a
        move                x:(r1+n1),b
        move    #-1,m0
        move    #-1,m1
        move    r0,x:dpleft
        move    r1,x:dpright

        JSR     STEREO_PROCESS

        ; MM 5/20/99 Swapped channels
        move    b,x:TX_BUFF_BASE+1          ;transmit left
        move    a,x:TX_BUFF_BASE+3          ;transmit right

        move    #>1,x0
        move    x:scnt,a
        add     x0,a
        move    a1,x:scnt               ; don't saturate

        jmp     LOOP


;---------------------------------------------------------------
; Subroutines
;---------------------------------------------------------------
STEREO_PROCESS

        move        a,x:left
        move        b,x:right

        ; copy parameters from left channel EQs to right channel EQs
        COPY_SH_PARAM   1L,1R
        COPY_SH_PARAM   2L,2R

        COPY_PEQ_PARAM  1L,1R
        COPY_PEQ_PARAM  2L,2R
        COPY_PEQ_PARAM  3L,3R
        COPY_PEQ_PARAM  4L,4R
        COPY_PEQ_PARAM  5L,5R
        COPY_PEQ_PARAM  6L,6R
        COPY_PEQ_PARAM  7L,7R
```

```
        COPY_PEQ_PARAM  NT1L,NT1R
        COPY_PEQ_PARAM  NT2L,NT2R
        COPY_PEQ_PARAM  NT3L,NT3R

        COPY_HP_PARAM   hpleft,hpright
        COPY_LP_PARAM   lpleft,lpright
        COPY_LP_PARAM   lpleft2,lpright2

        COPY_AP_PARAM   apleft,apright

        ; HDCD processing
;;      jset        #22,y:hdcdbp,doeq

        move        x:left,a
        move        x:right,b
        move        a,x:ltemp
        move        b,x:rtemp

        clr b
        move        #>1,x0
        move        y:gainscalebp,a
        tst         a
        teq         x0,b
        move        b1,x:dcontrol

        jsr         HDCD_DETECT

        ; destroy code
        move        #>3,x0
        move        x:rmodel,a
        cmp         x0,a
        bne         skip_code_dest
        move        #>8,x0
        move        x:bitcntl,a
        cmp         x0,a
        bne         skip_code_dest

        move        x:ltemp,a
        eor         #>$000100,a
        move        a,x:ltemp
        move        a,x:left

skip_code_dest:

        move        #-1,m0
        move        #-1,m4
        move        #4,n4
        jsr         HDCD_DYNAMICS

        move        x:ltemp,a
        move        x:rtemp,b
        btst        #22,y:hdcdbp
        .IF     <CC>
            move        a,x:left
            move        b,x:right
            move    x:detect,x0
            move    x0,x:spixmit
        .ELSE
            move    #0,x0
            move    x0,x:spixmit
        .ENDI
;       jmp         doeq1

;doeq:
;
;       move    #0,x0
;       move    x0,x:spixmit

doeq1:
        jset        #22,y:bypass,dopostvol

        move        x:left,x1
```

```
        move        x:right,y1
        move    y:prefader,x0
        mpyr    -x0,x1,a
        mpyr    -x0,y1,b

        tfr     a,b     b,x:right

        ; smiley face
        jset    #BYPASS_LOSHELV,y:bypassmask,_skiplsl
        LSH 1L
_skiplsl
        jset    #BYPASS_HISHELV,y:bypassmask,_skiphsl
        HSH 2L
_skiphsl
        ; cut-off response
        jset    #BYPASS_HIPASS,y:bypassmask,_skiphpl
        HP  hpleft
_skiphpl
        jset    #BYPASS_LOPASS,y:bypassmask,_skiplpl
        LP  lpleft
_skiplpl
        jset    #BYPASS_NLOPASS,y:bypassmask,_skiplp2l
        LP  lpleft2
_skiplp2l
        ; resonance EQ
        jset    #BYPASS_NOTCH1,y:bypassmask,_skip1l
        PEQ 1L
_skip1l
        jset    #BYPASS_NOTCH2,y:bypassmask,_skip2l
        PEQ 2L
_skip2l
        jset    #BYPASS_NOTCH3,y:bypassmask,_skip3l
        PEQ 6L
_skip3l
        jset    #BYPASS_NOTCH4,y:bypassmask,_skip4l
        PEQ.7L
_skip4l
        ; cone-cry
        jset    #BYPASS_CONECRY,y:bypassmask,_skipccl
        PEQ 5L
        PEQ 3L
        PEQ 4L
_skipccl
        ; double-tuned notch
        jset    #BYPASS_DBNOTCH,y:bypassmask,_skipdbnl
        PEQ NT1L
        PEQ NT2L
        PEQ NT3L
_skipdbnl
        ; all-pass
        jset    #BYPASS_ALLPASS,y:bypassmask,_skipapl
        AP  apleft
_skipapl

        move        b,x:left
        move        x:right,b

        ; smiley face
        jset    #BYPASS_LOSHELV,y:bypassmask,_skiplsr
        LSH 1R
_skiplsr
        jset    #BYPASS_HISHELV,y:bypassmask,_skiphsr
        HSH 2R
_skiphsr
        ; cut-off response
        jset    #BYPASS_HIPASS,y:bypassmask,_skiphpr
        HP  hpright
_skiphpr
        jset    #BYPASS_LOPASS,y:bypassmask,_skiplpr
        LP  lpright
_skiplpr
        jset    #BYPASS_NLOPASS,y:bypassmask,_skiplp2r
```

```
        LP  lpright2
_skiplp2r
        ; resonance EQ
        jset    #BYPASS_NOTCH1,y:bypassmask,_skip1r
        PEQ 1R
_skip1r
        jset    #BYPASS_NOTCH2,y:bypassmask,_skip2r
        PEQ 2R
_skip2r
        jset    #BYPASS_NOTCH3,y:bypassmask,_skip3r
        PEQ 6R
_skip3r
        jset    #BYPASS_NOTCH4,y:bypassmask,_skip4r
        PEQ 7R
_skip4r
        ; cone-cry
        jset    #BYPASS_CONECRY,y:bypassmask,_skipccr
        PEQ 5R
        PEQ 3R
        PEQ 4R
_skipccr
        ; double-tuned notch
        jset    #BYPASS_DBNOTCH,y:bypassmask,_skipdbnr
        PEQ NT1R
        PEQ NT2R
        PEQ NT3R
_skipdbnr
        ; all-pass
        jset    #BYPASS_ALLPASS,y:bypassmask,_skipapr
        AP  apright
_skipapr
        move        b,x:right

dopostvol:

plainvol:

        move    y:volume,y0
        move    x:left,x0
        move    x:right,x1
        mpyr    -x0,y0,a
        mpyr    -x1,y0,b
        rts
```

```
    org xi:




    org yi:

    org p:
```

```
;-------------------------------------------------------------
; SHI interrupts
;-------------------------------------------------------------

;   ; ---- SHI/I2C receive ISR
shi_rx_isr
        move    a1,x:shisaveA1
        move    x0,x:shisaveX0
        move    r4,x:shisaveR4
        move    n4,x:shisaveN4
        movep   x:M_HRX,a1
        jset    #SPIStateBit,x:flags,_gotData

        ; got index
        move    a1,x:indexData
        bset    #SPIStateBit,x:flags
        jmp     _exitInt

        ; got data
_gotData:
        move    x:indexData,n4
        move    #pptrs,r4
```

0135

```
        bclr    #SPIStateBit,x:flags
        movem   p:(r4+n4),r4
        move    a1,y:(r4)

_exitInt:
        move    x:shisaveA1,a1
        move    x:shisaveX0,x0
        move    x:shisaveR4,r4
        move    x:shisaveN4,n4
        rti

    ; ---- SHI/I2C receive overrun error
shi_rxe_isr
        movep   x:M_HCSR,x:NADA
        movep   x:M_HRX,x:NADA
        bset    #SPIOverrun,x:flags
        bclr    #SPIStateBit,x:flags
        rti

    ;   SHI Receive FIFO Full
shi_rxf_isr
        movep   x:M_HCSR,x:NADA
        movep   x:M_HRX,x:NADA
        bset    #SPIFIFOFull,x:flags
        bclr    #SPIStateBit,x:flags
        rti

     ;  SHI Transmit Data
shi_txu_isr                         ;SHI Transmit Underrun Error
        bset    #SPIUnderrun,x:flags
        movep   x:M_HCSR,x:NADA
        movep   x:spixmit,x:M_HTX
        bclr    #SPIStateBit,x:flags
        rti

shi_tx_isr:
        movep   x:spixmit,x:M_HTX
        bset    #SPITransmit,x:flags
        rti

       ;SHI Bus Error
shi_bus_error:
;       movep   x:M_HCSR,x:NADA
;       bset    #SPIReceive+4,x:flags
        ; reset shi
        bclr    #0,x:M_HCSR
        nop
        nop
        bset    #0,x:M_HCSR
        bclr    #SPIStateBit,x:flags
        nop
        rti

;------------------------------------------------------------
; Subroutines
;------------------------------------------------------------
;               include 'isr_dig.asm'

;------------------------------------------------------------
; Interrupt Service Routines
;------------------------------------------------------------

esai_txe_isr                            ; ESAI TRANSMIT ISR
        bclr    #14,x:M_SAISR           ; Read SAISR to clear transmit
                                        ; underrun error flag
esai_tx_isr

        jset    #13,x:M_SAISR,TxLeftSlot

        movep   x:TX_BUFF_BASE+2,x:M_TX1            ; write unprocessed data
        movep   x:TX_BUFF_BASE+3,x:M_TX0
        movep   x:TX_BUFF_BASE+3,x:M_TX2
```

0136

```
        movep   x:TX_BUFF_BASE+3,x:M_TX3
        rti

TxLeftSlot

        movep   x:TX_BUFF_BASE,x:M_TX1          ; write unprocessed data
        movep   x:TX_BUFF_BASE+1,x:M_TX0
        movep   x:TX_BUFF_BASE+1,x:M_TX2
        movep   x:TX_BUFF_BASE+1,x:M_TX3
        rti

esai_txls_isr                               ; ESAI TRANSMIT LAST SLOT ISR
;        move    r0,x:(r6)+                  ; Save r0 to the stack
;        move    #TX_BUFF_BASE,r0            ; Reset pointer
;        move    r0,x:TX_PTR                 ; Reset tx buffer pointer just in
;                                            ; case it was corrupted
;        move    x:-(r6),r0                  ; Restore r0
        rti

esai_rxe_isr                                ; ESAI RECEIVE ISR
        bclr    #7,x:M_SAISR                ; Read SAISR to clear receive
 overrun error flag
                                            ; overrun error flag
esai_rx_isr

        jset    #$6,x:M_SAISR,LeftSlot
        bset    #RightReceive,x:flags  ; if right channel data then set flag
        movep   x:M_RX0,x:ARX_BUFF_BASE+1
        movep   x:M_RX1,x:DRX_BUFF_BASE+1
        rti

LeftSlot
        movep   x:M_RX0,x:ARX_BUFF_BASE
        movep   x:M_RX1,x:DRX_BUFF_BASE
        rti

esai_rxls_isr                               ; ESAI RECEIVE LAST SLOT ISR
;        move    r0,x:(r6)+                  ; Save r0 to the stack
;       move    #RX_BUFF_BASE,r0            ; Reset rx buffer pointer just in
;                                            ; case it was corrupted
;        move    r0,x:RX_PTR                 ; Update rx buffer pointer
;        move    x:-(r6),r0                  ; Restore r0
        rti

;   variable look up table

pptrs
    dc  volume
    dc  sh_gamma_1L
    dc  sh_k_1L
    dc  sh_gamma_2L
    dc  sh_k_2L
    dc  peq_gamma_1L
    dc  peq_beta_1L
    dc  peq_k_1L
    dc  peq_gamma_2L
    dc  peq_beta_2L
    dc  peq_k_2L
    dc  peq_gamma_3L
    dc  peq_beta_3L
    dc  peq_k_3L
    dc  peq_gamma_4L
    dc  peq_beta_4L
    dc  peq_k_4L
    dc  peq_gamma_5L
    dc  peq_beta_5L
    dc  peq_k_5L
    dc  prefader
    dc  bypass
    dc  hp_scale_hpleft
    dc  hp_fc_hpleft
    dc  hp_qc_hpleft
```

```
        dc  lp_scale_lpleft
        dc  lp_a2_lpleft
        dc  lp_a1_lpleft
        dc  hdcdbp
        dc  gainscalebp
        dc  ddxcompbp
        dc  peq_gamma_6L
        dc  peq_beta_6L
        dc  peq_k_6L
        dc  peq_gamma_7L
        dc  peq_beta_7L
        dc  peq_k_7L
        dc  analogin
        dc  avolume
        dc  bypassmask
        dc  delayval
        dc  ap_gamma_apleft
        dc  ap_beta_apleft
        dc  lp_scale_lpleft2
        dc  lp_a2_lpleft2
        dc  lp_a1_lpleft2
;       dc  lp_b2_lpleft2
;       dc  lp_b1_lpleft2
        dc  peq_gamma_NT1L
        dc  peq_beta_NT1L
        dc  peq_k_NT1L
        dc  peq_gamma_NT2L
        dc  peq_beta_NT2L
        dc  peq_k_NT2L
        dc  peq_gamma_NT3L
        dc  peq_beta_NT3L
        dc  peq_k_NT3L
        dc  YNADA
        dc  YNADA
        dc  YNADA


            ENDSEC


            end
```

```
SH_SCALE_FACTOR EQU 3
SH_SCALE_DIVIDE EQU (1<<SH_SCALE_FACTOR)

;
;   k is gain
;       k = 0       => lo/hi pass filter
;       0 < k < 1   => cut
;       k = 1       => pass-thru
;       k > 1       => boost
;
;   gamma drives critical frequency
;
;

DECLARE_LSH MACRO       name,k,gamma

    org xi:
sh_x_\name          dc  0           ; x(n-1)
sh_y_\name          dc  0           ; y(n-1)

    org yi:
sh_gamma_\name      dc  gamma
sh_minus_one_\name  dc  -1.0
sh_k_\name          dc  k/SH_SCALE_DIVIDE
sh_1_\name          dc  1./SH_SCALE_DIVIDE

    ENDM

DECLARE_HSH MACRO       name,k,gamma

    DECLARE_LSH name,k,gamma

    ENDM

COPY_SH_PARAM   MACRO       from,to

    move    y:sh_gamma_\from,x0
    move    x0,y:sh_gamma_\to
    move    y:sh_k_\from,x0
    move    x0,y:sh_k_\to

    ENDM

INIT_SH_PARAM   MACRO       name

    move    #>(-1.0),x0
    move    x0,y:sh_minus_one_\name
    move    #>(1./SH_SCALE_DIVIDE),x0
    move    x0,y:sh_k_\name
    move    x0,y:sh_1_\name

    ENDM


;
;       input data is in x0
;       input is scaled by 0.5 to prevent internal clipping in the all-pass
;
;       computes all-pass:
;
;       y(n) = - gamma * x(n) - x(n-1) - gamma * y(n-1)  (lo-shelv)
;       y(n) = gamma * x(n) + x(n-1) - gamma * y(n-1)  (hi-shelv)
;
;       all-pass output is scaled by gain:
;
;       output = ((1+k)/2) * x(n) + ((1-k)/2) * y(n)
;
;       assumes:
;           m0,m4,m5 = -1
;
;
```

```
LSH MACRO       name

    asr     b               #sh_gamma_\name,r4
    rnd     b               #sh_x_\name,r0
    move                    b,x0            y:(r4)+,y0
    ; compute y(n)
    mpy     -x0,y0,b        x:(r0)+,x1      y:(r4)+,y1
    mac     x1,y1,b         x:(r0)-,x1
    macr    -y0,x1,b        x0,x:(r0)+      y:(r4)+,y0
    ; all-pass output in b, now scale by gain/cut factor
    mpy     x0,y0,b         b,x:(r0)-       b,y1
    mac     -y1,y0,b                        y:(r4)+,y0
    mac     x0,y0,b
    mac     y1,y0,b
    ; now scale output to get real result
    asl     #(SH_SCALE_FACTOR),b,b
    rnd     b

    ENDM

HSH MACRO       name

    asr     b               #sh_gamma_\name,r4
    rnd     b               #sh_x_\name,r0
    move                    b,x0            y:(r4)+,y0
    ; compute y(n)
    mpy     x0,y0,b         x:(r0)+,x1      y:(r4)+,y1
    mac     -x1,y1,b        x:(r0)-,x1
    macr    -y0,x1,b        x0,x:(r0)+      y:(r4)+,y0
    ; all-pass output in b, now scale by gain/cut factor
    mpy     x0,y0,b         b,x:(r0)-       b,y1
    mac     -y1,y0,b                        y:(r4)+,y0
    mac     x0,y0,b
    mac     y1,y0,b
    ; now scale output to get real result
    asl     #(SH_SCALE_FACTOR),b,b
    rnd     b

    ENDM
```

```
PEQ_SCALE_FACTOR    EQU 3
PEQ_SCALE_DIVIDE    EQU (1<<PEQ_SCALE_FACTOR)

;
;   k is gain
;       k = 0       => notch filter
;       0 < k < 1   => cut
;       k = 1       => pass-thru
;       k > 1       => boost
;
;   gamma drives critical frequency
;
;   beta defines Q
;

DECLARE_PEQ MACRO       name,k,gamma,beta

    org xi:
peq_y_\name         dc  0
                    dc  0
peq_x_\name         dc  0
                    dc  0

    org yi:
peq_gamma_\name     dc  gamma
peq_beta_\name      dc  beta
peq_k_\name         dc  k/PEQ_SCALE_DIVIDE
peq_1_\name         dc  1./PEQ_SCALE_DIVIDE
peq_v_\name         dc  0
peq_w_\name         dc  0

    ENDM

COPY_PEQ_PARAM  MACRO       from,to

    move    y:peq_gamma_\from,x0
    move    x0,y:peq_gamma_\to
    move    y:peq_beta_\from,x0
    move    x0,y:peq_beta_\to
    move    y:peq_k_\from,x0
    move    x0,y:peq_k_\to

    ENDM

INIT_PEQ_PARAM  MACRO       name

    move    #>(1./PEQ_SCALE_DIVIDE),x0
    move    x0,y:peq_k_\name
    move    x0,y:peq_1_\name

    ENDM

;
;       input data is in b
;       input is scaled by 0.5 to prevent internal clipping in the all-pass
;
;       computes all-pass:
;
;       v(n) = gamma * y(n-1) + y(n-2) - gamma * v(n-1)
;       w(n) = gamma * x(n-1) + x(n-2) - gamma * w(n-1)
;       y(n) = w(n) + beta * x(n) - beta * v(n)
;
;       all-pass output is scaled by gain:
;
;       output = ((1+k)/2) * x(n) + ((1-k)/2) * y(n)
;
;       assumes:
;           m0,m4,m5 = -1
;
;
```

0141

```
PEQ MACRO       name

    asr     b       #peq_y_\name,r0
    move            #peq_gamma_\name,r4
    move            #peq_v_\name,r5
    move                    b,x0
    move                    x:(r0)+,a       y:(r4)+,y0
    move                    x:(r0)+,x1      y:(r5)+,y1
    ; compute v(n)
    mac     x1,y0,a         x:(r0)+,b
    mac     -y1,y0,a        x:(r0),x1       y:(r5)-,y1
    ; compute w(n), v(n) is now in a
    mac     x1,y0,b         a,x1            a,y:(r5)+
    mac     -y1,y0,b        x:(r0),a        y:(r4)+,y0
    ; compute y(n), w(n) is now in b
    mac     -x1,y0,b        x0,x:(r0)-      b,y:(r5)-
    mac     x0,y0,b         a,x:(r0)-       y:(r4)+,y0
    ; all-pass output in b, now scale by gain/cut factor
    mpy     x0,y0,b         x:(r0)-,a       b,y1
    mac     -y1,y0,b                        y:(r4)+,y0
    mac     x0,y0,b         a,x:(r0)+
    mac     y1,y0,b         y1,x:(r0)-
    ; now scale output to get real result
    asl     #(PEQ_SCALE_FACTOR),b,b
    rnd     b

    ENDM
```

```
LP_SCALE_FACTOR EQU 6
LP_SCALE_DIVIDE EQU (1<<LP_SCALE_FACTOR)

;
;
;   modified chamberlin lo-pass (both x(n) and x(n-1))
;
;       q also defines gain
;       input is scaled to avoid internal clipping
;
;   f = 2*sin(w/2)  f only valid when less than 1.
;
;   0 < q < 1
;
;   implemented as a direct form biquad with coefficients and data scaled.
;   needs headroom for gain
;

DECLARE_LP  MACRO       name,ca1,ca2,cb

    org xi:
lp_w_\name          ds  2

    org yi:
lp_scale_\name      dc  cb/16.
lp_a2_\name         dc  ca2/2.
lp_a1_\name         dc  ca1/2.
lp_b2_\name         dc  0.0
lp_b1_\name         dc  1.0/2.

    ENDM


COPY_LP_PARAM   MACRO       from,to

    move    y:lp_scale_\from,x0
    move    x0,y:lp_scale_\to
    move    y:lp_a2_\from,x0
    move    x0,y:lp_a2_\to
    move    y:lp_a1_\from,x0
    move    x0,y:lp_a1_\to
    move    y:lp_b2_\from,x0
    move    x0,y:lp_b2_\to
    move    y:lp_b1_\from,x0
    move    x0,y:lp_b1_\to

    ENDM

INIT_LP_PARAM   MACRO       name

    move    #0,x0
    move    x0,y:lp_b2_\name
    move    x0,y:lp_a2_\name
    move    x0,y:lp_a1_\name
    move    x0,y:lp_b1_\name
    move    #>(1./16.),x0
    move    x0,y:lp_scale_\name

    ENDM

INIT_LP2_PARAM  MACRO       name

    move    #0,x0
    move    x0,y:lp_a2_\name
    move    x0,y:lp_a1_\name
    move    #>0.5,x0
    move    x0,y:lp_b2_\name
    move    #>1.,x0
    move    x0,y:lp_b1_\name
```

```
        move    #>(0.25/16.),x0
        move    x0,y:lp_scale_\name

        ENDM


;
;       assumes:
;           m0,m4,m5 = -1
;
;

LP  MACRO       name

        move    b,x0
        ori     #8,mr
        move    #lp_scale_\name,r4
        move    #lp_w_\name,r0
        move                                        y:(r4)+,y0      ; x0 = x(n), y0 = 0.5
        mpy     x0,y0,b     x:(r0)+,x0              y:(r4)+,y0      ; x0 = w(n-2), y0 = a2
        mac     -x0,y0,b    x:(r0)-,x1              y:(r4)+,y0      ; x1 = W(n-1), y0 = a1
        macr    -x1,y0,b    x1,x:(r0)+              y:(r4)+,y0      ; w(n-2) = w(n-1), y0 = b2
        mac     x0,y0,b     b,x:(r0)-               y:(r4)+,y0      ; w(n-1) = a, y0 = b1
        macr    x1,y0,b
        andi    #$F7,mr
        asl     #4,b,b

        ENDM
```

```
HP_SCALE_FACTOR EQU 6
HP_SCALE_DIVIDE EQU (1<<HP_SCALE_FACTOR)

;
;
;   chamberlin hi-pass (from lopass)
;
;       q also defines gain
;       input is scaled to avoid internal clipping
;
;   f = 2*sin(w/2)  f only valid when less than 1.
;
;   0 < q < 1
;
;   implemented as a direct form biquad with coefficients and data scaled.
;   needs headroom for gain
;

DECLARE_HP  MACRO       name,fc,qc,scale

    org xi:
hp_s2_\name         ds  1
hp_y_\name          ds  1

    org yi:
hp_scale_\name      dc  (-0.5*scale)
hp_fc_\name         dc  fc
hp_qc_\name         dc  qc

    ENDM


COPY_HP_PARAM   MACRO       from,to

    move    y:hp_scale_\from,x0
    move    x0,y:hp_scale_\to
    move    y:hp_fc_\from,x0
    move    x0,y:hp_fc_\to
    move    y:hp_qc_\from,x0
    move    x0,y:hp_qc_\to


    ENDM


;
;       assumes:
;           m0,m4,m5 = -1
;
;

HP  MACRO       name

    move    #hp_scale_\name,r4
    move    #hp_s2_\name,r0
    move                b,x0            y:(r4)+,y0      ; y0 = scale
    ; a = x(n) * scale
    mpy     -y0,x0,a    x:(r0)+,x1      y:(r4)+,y0      ; y0 = fc
    ; b = s2(n-1) * fc
    mpyr    x1,y0,b     x:(r0),x0       y:(r4)+,y1      ; y1 = qc
    ; b = s2(n-1) * fc + y(n-1)
    add     x0,b
    ; a = x(n) - s2(n-1) * fc - y(n-1)
    sub     b,a         b,x:(r0)
    ; a = s1(n) = x(n) - s2(n-1) * fc - y(n-1) - qc * s2(n-1)
    macr    -x1,y1,a    x1,b
    move                a,x0
    ; b = s2(n) = s2(n-1) + fc * s1(n)
    macr    x0,y0,b     x:(r0)-,x0
```

```
IF  @DEF('LO')
tfr     x0,b        b,x:(r0)
ELSE
tfr     a,b         b,x:(r0)
ENDIF
asl     b

ENDM
```

```
AP_SCALE_FACTOR EQU 3
AP_SCALE_DIVIDE EQU (1<<AP_SCALE_FACTOR)

;
;   gamma drives critical frequency
;
;   beta defines Q
;

DECLARE_AP  MACRO       name,gamma,beta

    org xi:
ap_y_\name          dc  0
                    dc  0
ap_x_\name          dc  0
                    dc  0

    org yi:
ap_gamma_\name      dc  gamma
ap_beta_\name       dc  beta
ap_v_\name          dc  0
ap_w_\name          dc  0

    ENDM

COPY_AP_PARAM   MACRO       from,to

    move    y:ap_gamma_\from,x0
    move    x0,y:ap_gamma_\to
    move    y:ap_beta_\from,x0
    move    x0,y:ap_beta_\to

    ENDM

INIT_AP_PARAM   MACRO       name

    ENDM

;
;       input data is in b
;       input is scaled by 0.5 to prevent internal clipping in the all-pass
;
;       computes all-pass:
;
;       v(n) = gamma * y(n-1) + y(n-2) - gamma * v(n-1)
;       w(n) = gamma * x(n-1) + x(n-2) - gamma * w(n-1)
;       y(n) = w(n) + beta * x(n) - beta * v(n)
;
;       assumes:
;           m0,m4,m5 = -1
;
;

AP  MACRO       name

    asr     b       #ap_y_\name,r0
    move            #ap_gamma_\name,r4
    move            #ap_v_\name,r5
    move                    b,x0
    move                    x:(r0)+,a       y:(r4)+,y0
    move                    x:(r0)+,x1      y:(r5)+,y1
    ; compute v(n)
    mac     x1,y0,a         x:(r0)+,b
    mac     -y1,y0,a        x:(r0),x1       y:(r5)-,y1
    ; compute w(n), v(n) is now in a
    mac     x1,y0,b         a,x1            a,y:(r5)+
    mac     -y1,y0,b        x:(r0),a        y:(r4)+,y0
    ; compute y(n), w(n) is now in b
    mac     -x1,y0,b        x0,x:(r0)-      b,y:(r5)-
    mac     x0,y0,b         a,x:(r0)-
```

```
asl     b               x:(r0)-,a       b,y1
rnd     b               a,x:(r0)+
move                    y1,x:(r0)-

ENDM
```

include